



## Deep Blue versus Garry Kasparov



1997 chess match between world champion Garry Kasparov and IBM computer "Deep Blue"



# AlphaGo versus Lee Sedol

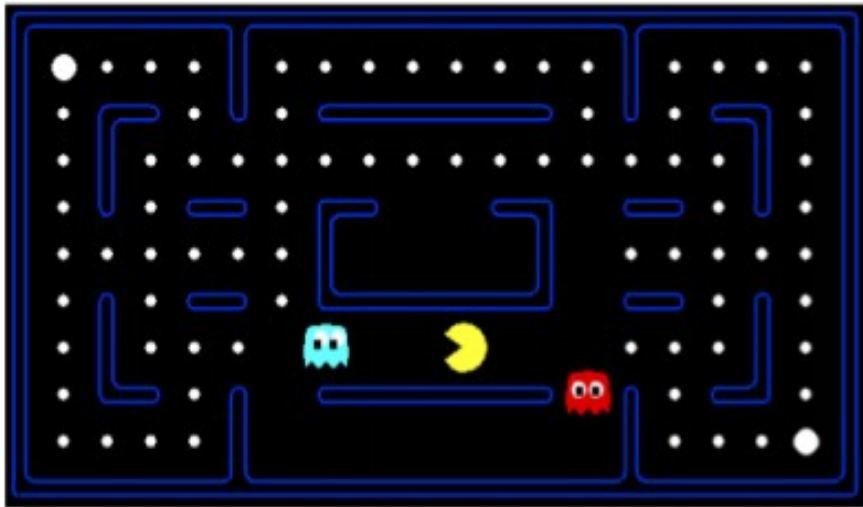




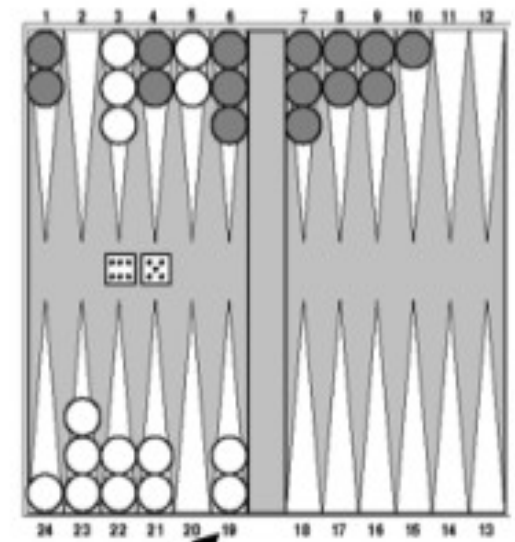
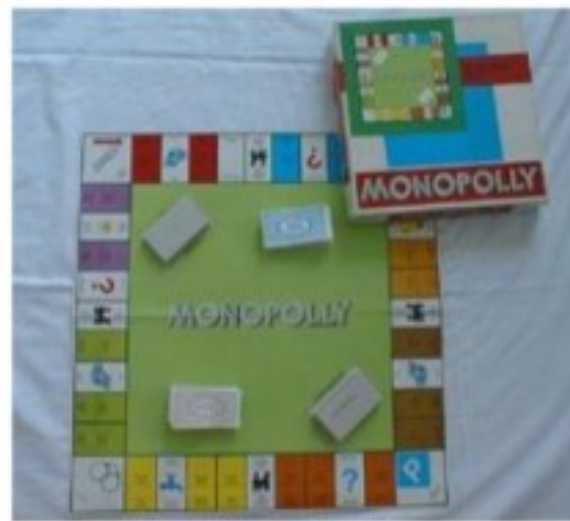


# Adversarial Search and Game playing.

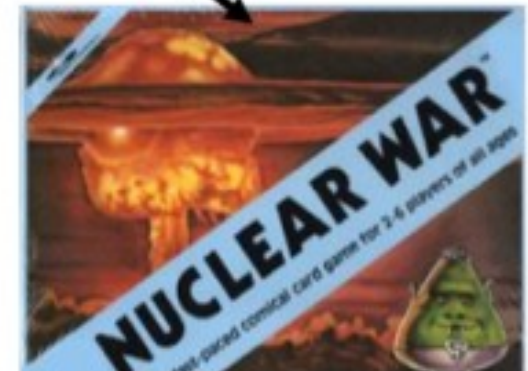
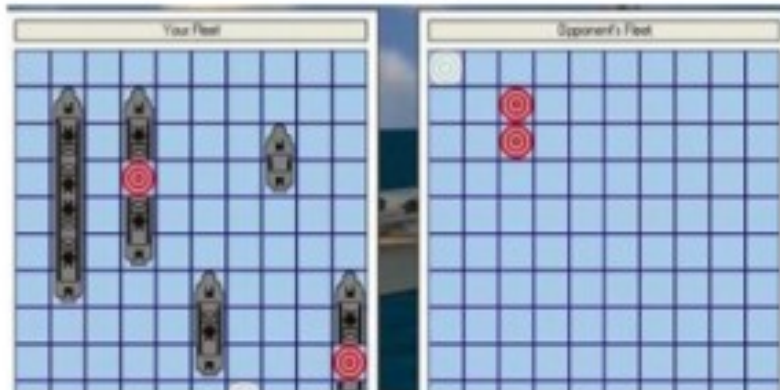
- ✓ Competitive environments in which the agents goals are in conflict, give rise to adversarial search, often known as games.



# Type of games



	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war





# Game

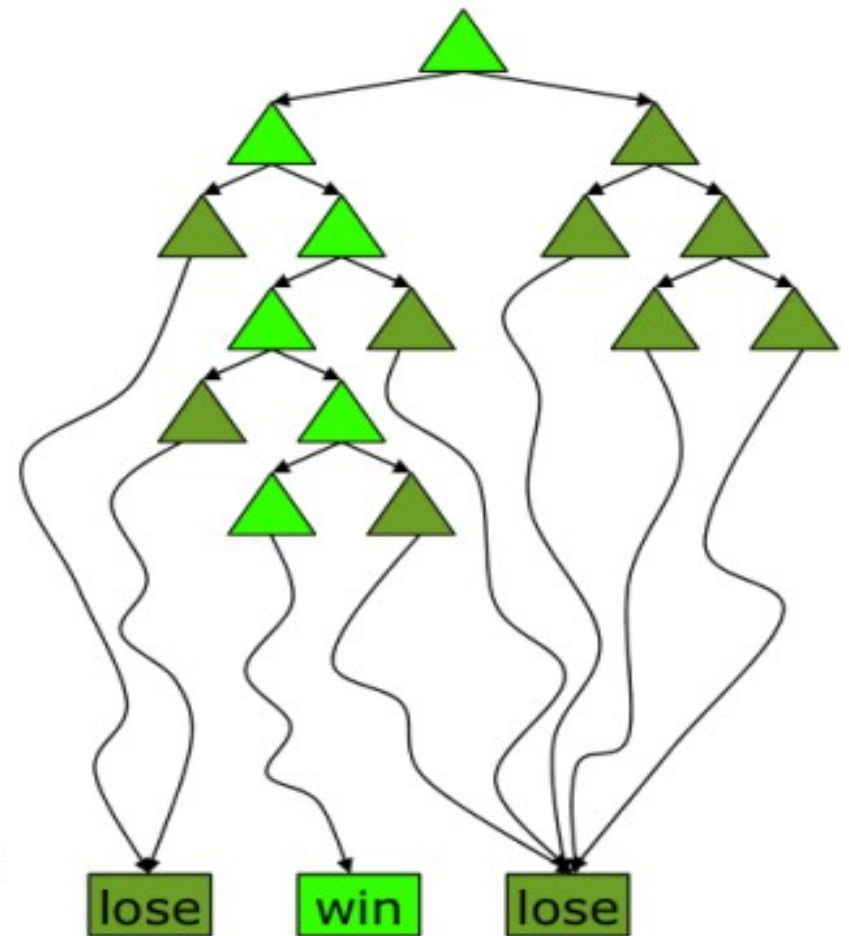
- ✓ A game can be formally be defined as a kind of search problem with the following components.
  - Initial state
  - A successor function
  - A terminal test
  - A utility function



# Game Playing

## Deterministic Single-Player?

- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- ... it's just search!
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node



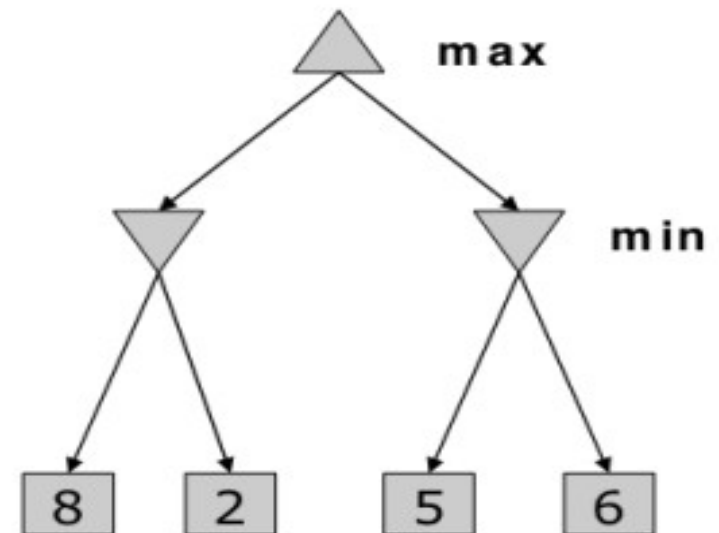




# Game Playing

## Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
- Minimax search
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves\*
  - Choose move to position with highest minimax value = best achievable utility against best play



*\* Slightly different from the book definition*



# Game Playing

## Games vs. search problems

- "Unpredictable" opponent → solution is a strategy specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate
- Plan of attack:
  - Computer considers possible lines of play (Babbage, 1846)
  - Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
  - Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
  - First chess program (Turing, 1951)
  - Machine learning to improve evaluation accuracy (Samuel, 1952-57)
  - Pruning to allow deeper search (McCarthy, 1956)





# Game Playing

## Searching for the next move

- **Complexity:** many games have a huge search space
  - **Chess:**  $b = 35, m=100 \Rightarrow \text{nodes} = 35^{100}$   
if each node takes about 1 ns to explore  
then each move will take about  **$10^{50}$  millennia** to calculate.
- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate
  1. **Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality result.
  2. **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.



# Game Playing

## Two-player games

- A game formulated as a search problem:
  - Initial state: board position and turn
  - Operators: definition of legal moves
  - Terminal state: conditions for when game is over
  - Utility function: a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win. (AKA **payoff function**)



# Minimax Algorithm

- ✓ Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.
- ✓ It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Chess, etc.





# Minimax Algorithm

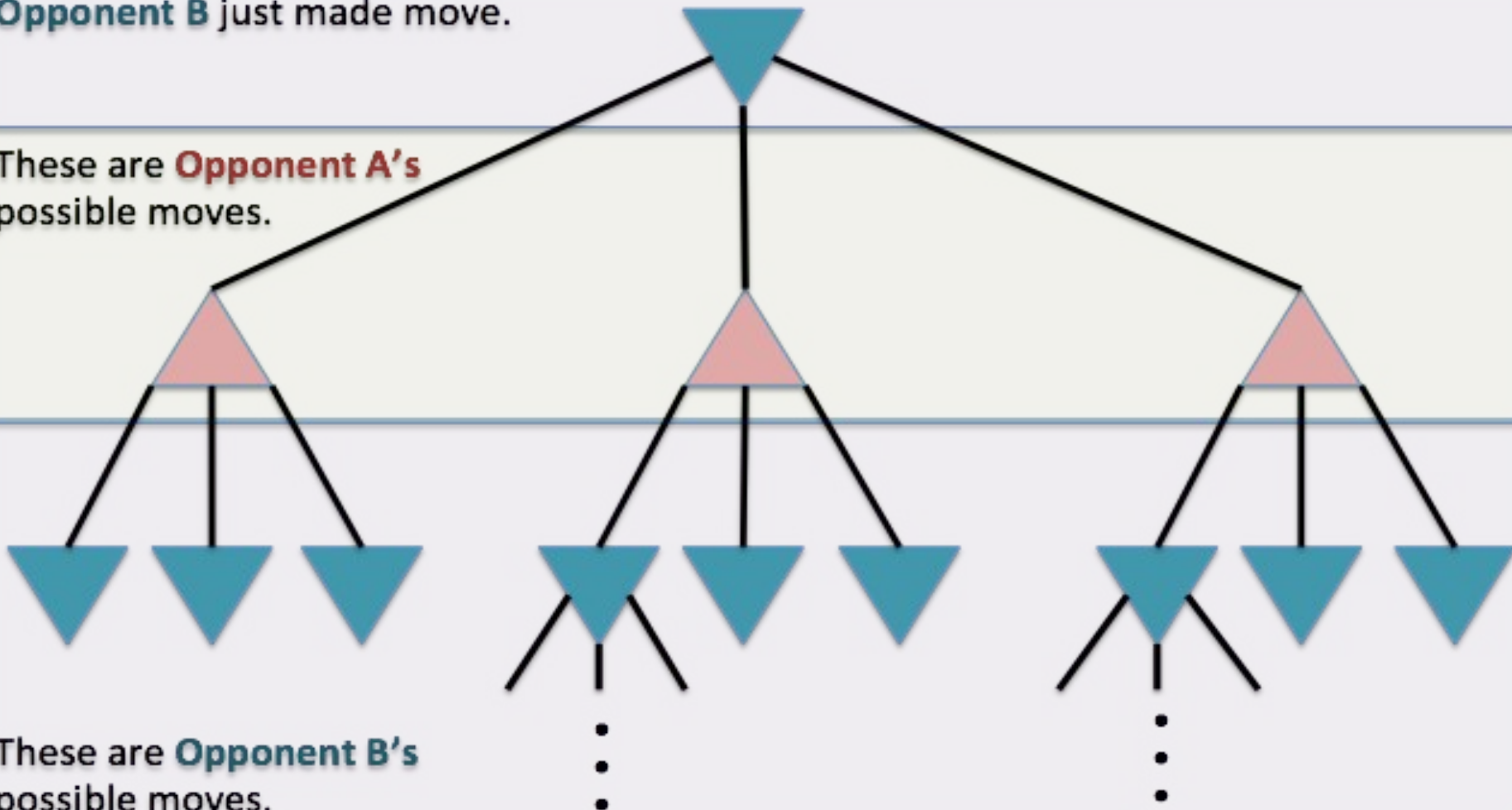
- ✓ In Minimax the two players are called maximizer and minimizer.
- ✓ The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.



# Minimax Algorithm

**Opponent B** just made move.

These are **Opponent A's** possible moves.



These are **Opponent B's** possible moves.



# Minimax Algorithm

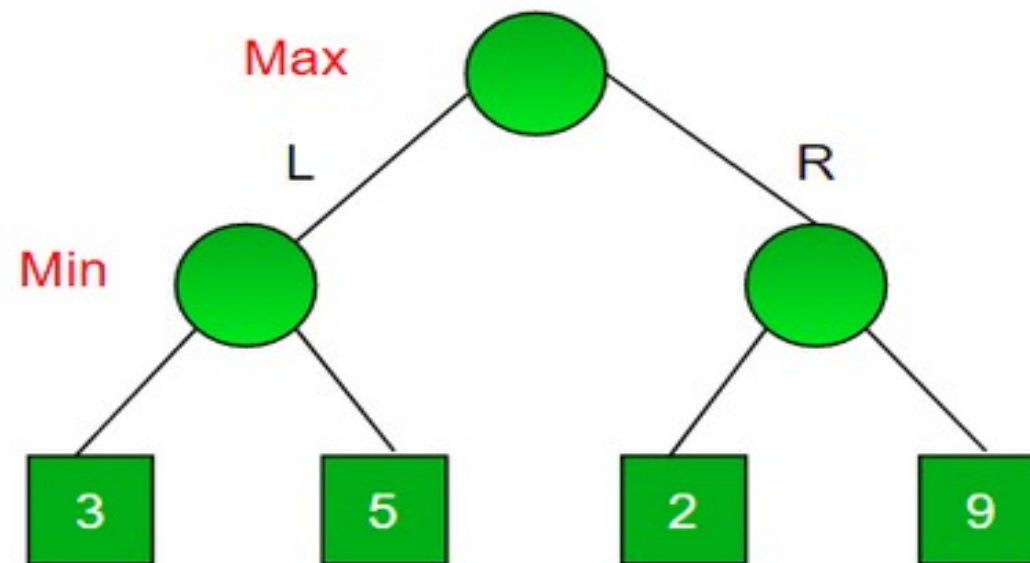
- ✓ Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value.
- ✓ If the minimizer has the upper hand in that board state then it will tend to be some negative value.
- ✓ The values of the board are calculated by some heuristics which are unique for every type of game.





# Minimax Algorithm

- ✓ Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level. **Which move you would make as a maximizing player considering that your opponent also plays optimally?**





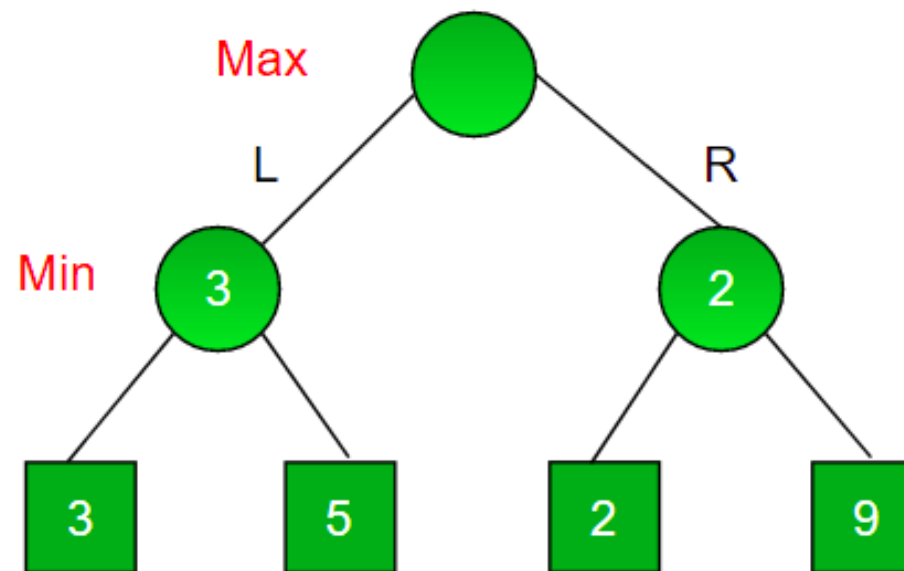
# Minimax Algorithm

- ✓ Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
- ✓ Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.



# Minimax Algorithm

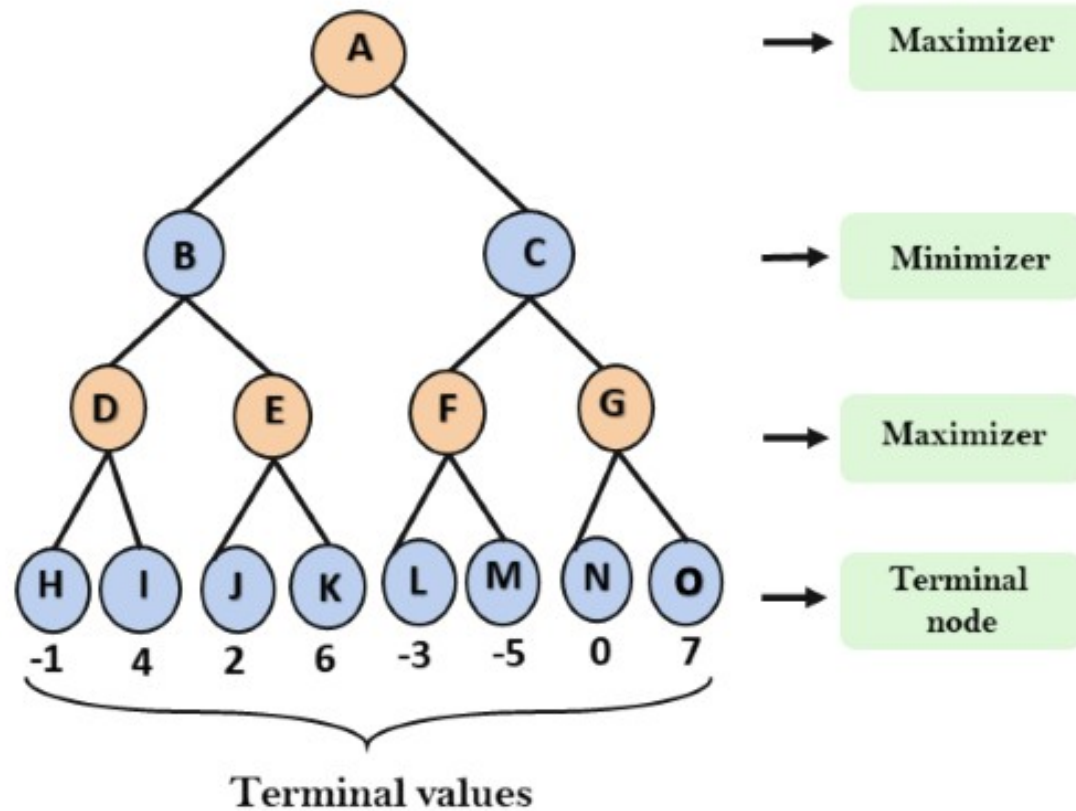
- ✓ Being the maximizer you would choose the larger value that is 3.
- ✓ Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.





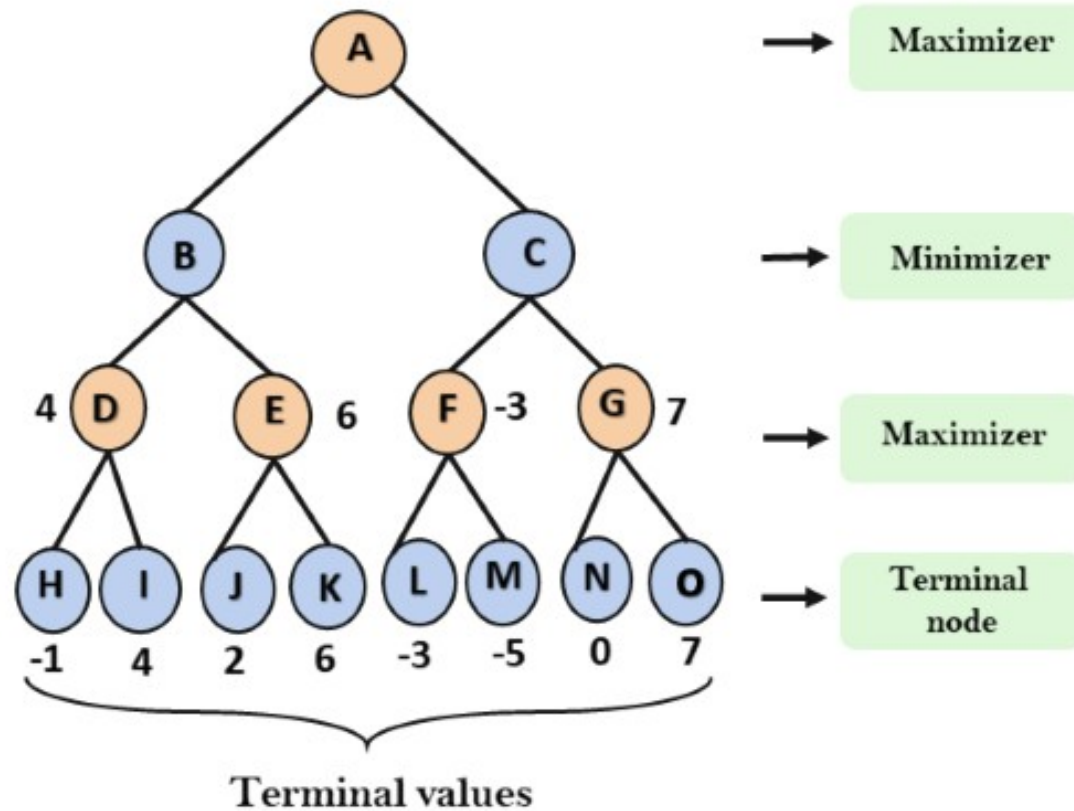


# Minimax Algorithm



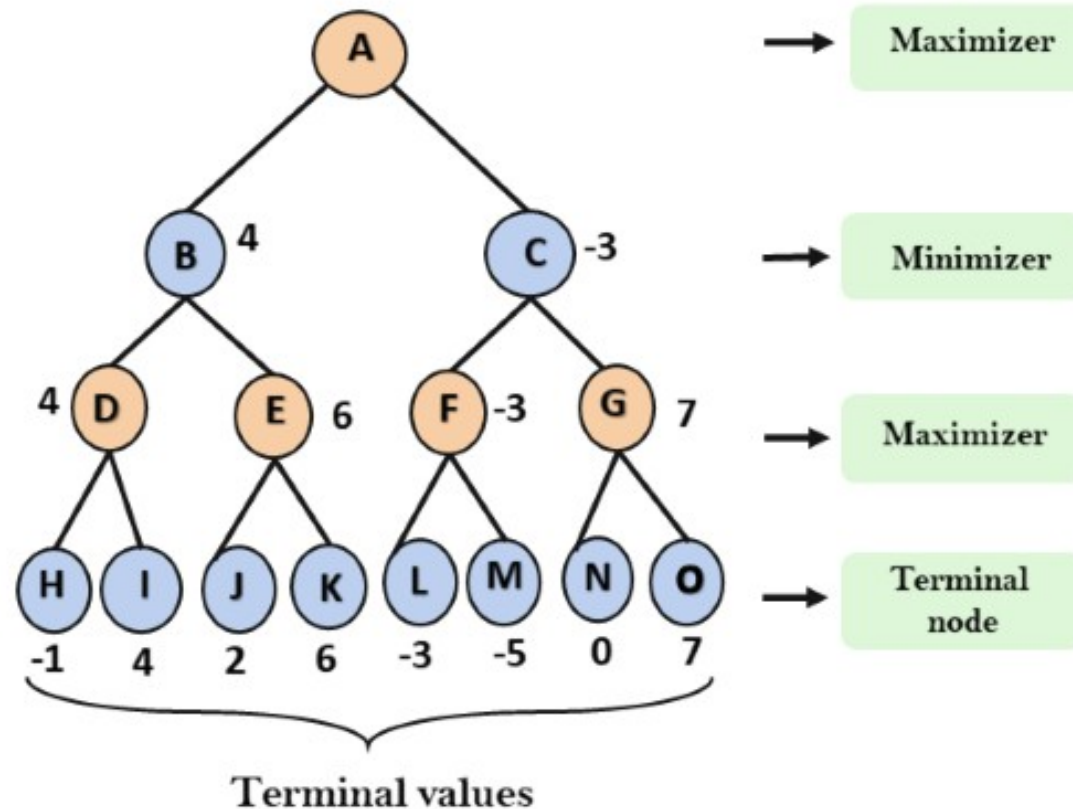


# Minimax Algorithm





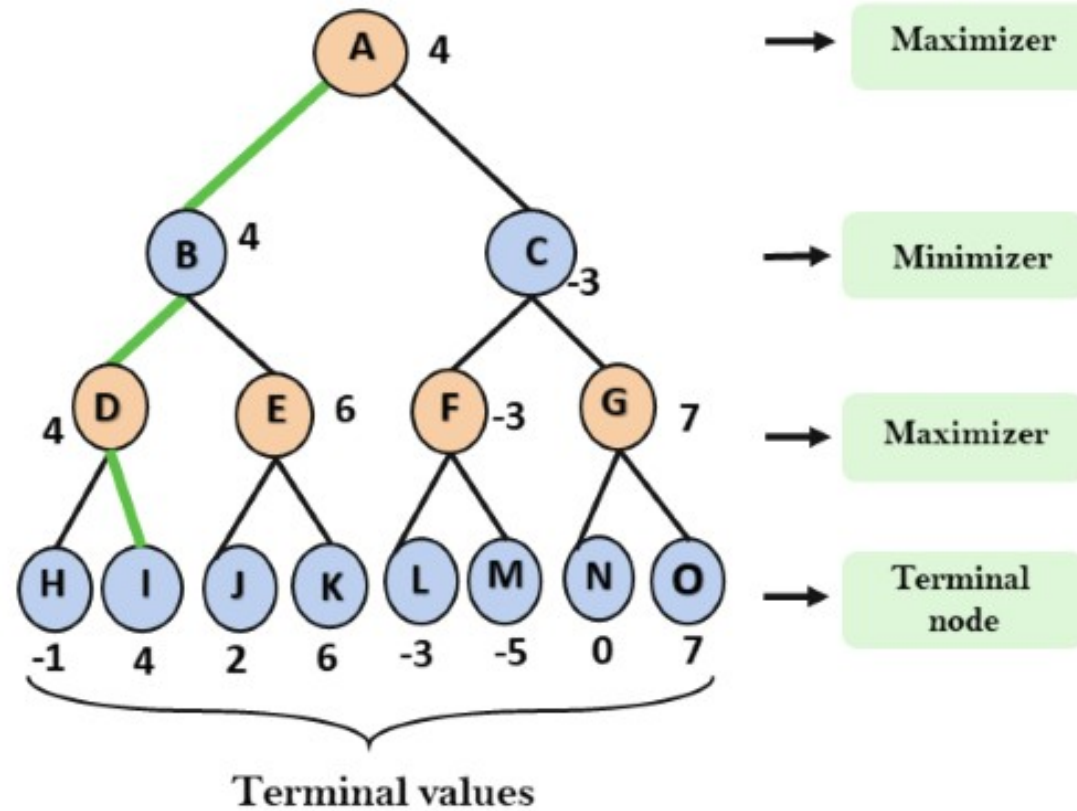
# Minimax Algorithm







# Minimax Algorithm





# Minimax Algorithm

- ✓ In the real world when we are creating a program to play Tic-Tac-Toe, Chess, Backgammon, etc. we need to implement a function that calculates the value of the board depending on the placement of pieces on the board.
- ✓ This function is often known as Evaluation Function.
- ✓ It is sometimes also called Heuristic Function.



# Minimax Algorithm

- ✓ The basic idea behind the evaluation function is to give a high value for a board if maximizer's turn or a low value for the board if minimizer's turn.
- ✓ For example let us consider X as the maximizer and O as the minimizer.



# Minimax Algorithm

- ✓ If X wins on the board we give it a positive value of +10.

X	0	0
	X	
		X

+10



# Minimax Algorithm

- ✓ If O wins on the board we give it a negative value of -10

0	0	0
	X	X
		X
-10		



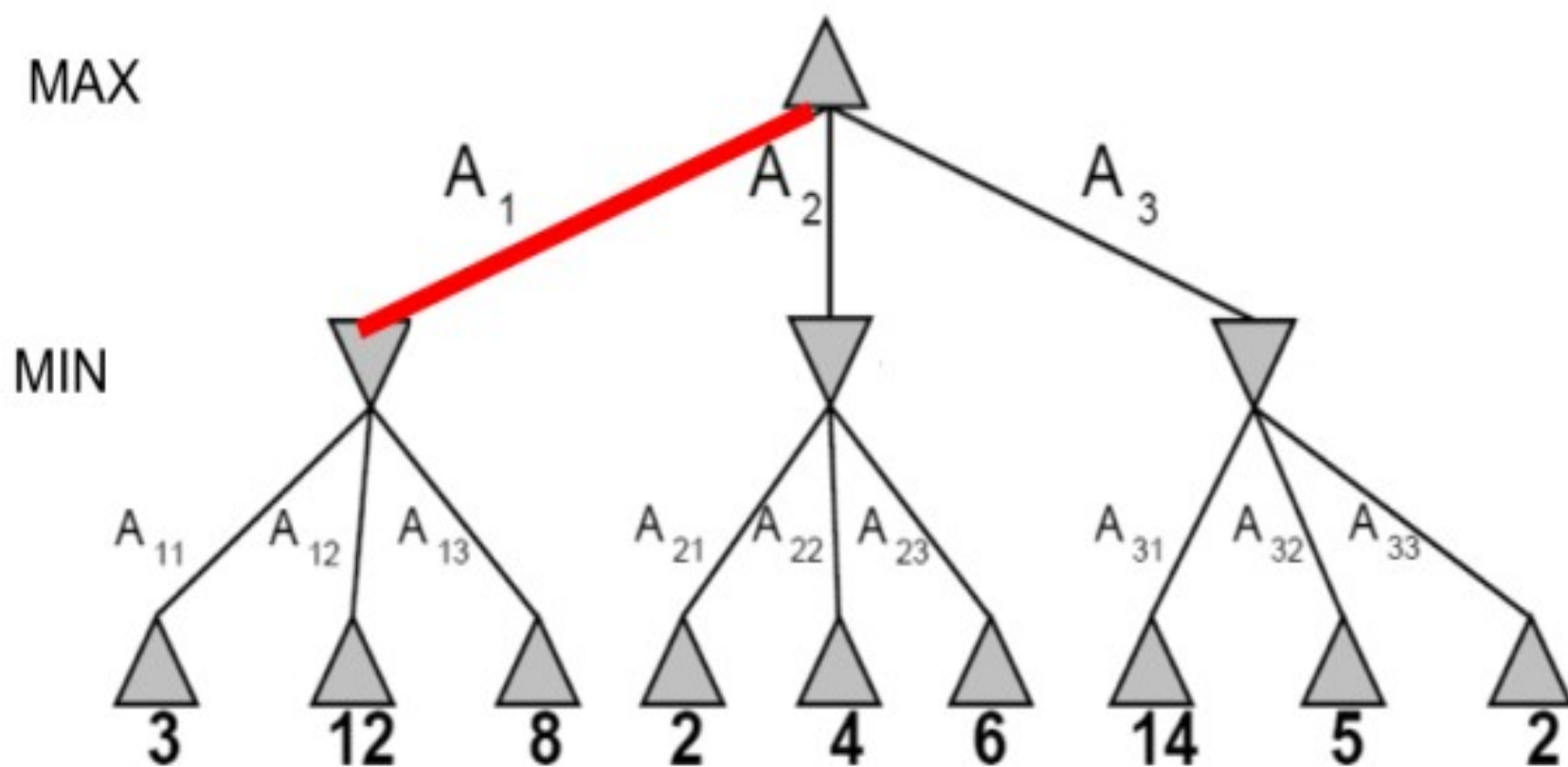


# Minimax Algorithm

- ✓ If no one has won or the game results in a draw then we give a value of +0.

X	0	X
0	X	X
0	X +0	0

# Minimax Example



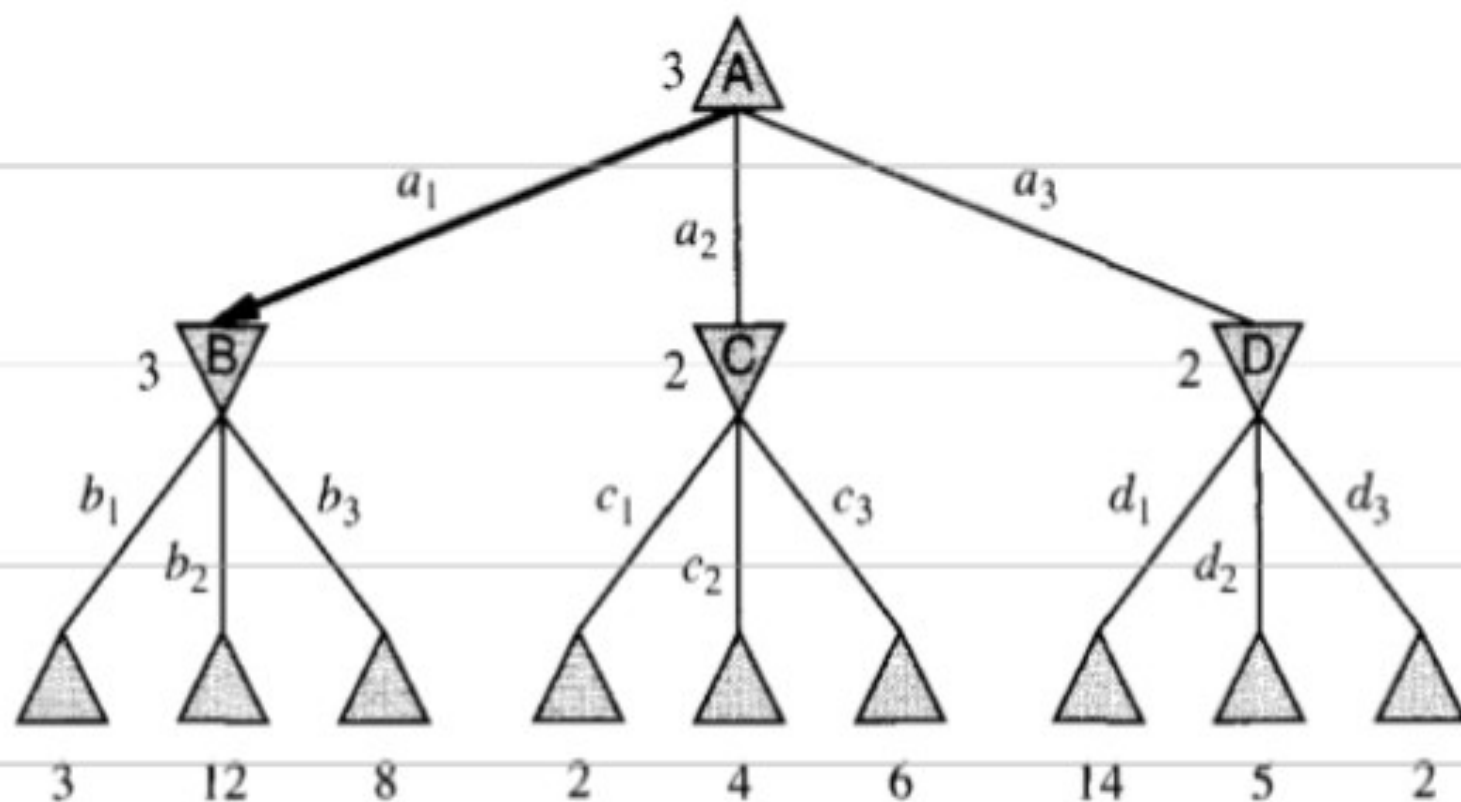
# Minimax Example

MINIMAX-VALUE( $n$ ) =

$$\begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{Successors}(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Successors}(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

MAX

MIN





## Properties of Minimax algorithm:

- **Complete**
  - Minimax algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal**
  - Minimax algorithm is optimal if both opponents are playing optimally.
- **Time complexity**
  - As it performs DFS for the game-tree, so the time complexity of Minimax algorithm is  $O(b^m)$ , where  $b$  is branching factor of the game-tree, and  $m$  is the maximum depth of the tree.
- **Space Complexity**
  - Space complexity of Mini-max algorithm is also similar to DFS which is  $O(bm)$ .



## Limitation of the Minimax Algorithm

- ✓ The main drawback of the Minimax algorithm is that it gets really slow for complex games such as Chess, go, etc.
- ✓ This type of games has a huge branching factor, and the player has lots of choices to decide.
- ✓ This limitation of the Minimax algorithm can be improved from **alpha-beta pruning**.





## Alpha-Beta Pruning.

- ✓ Alpha-beta pruning is a modified version of the Minimax algorithm
- ✓ It is an optimization technique for the Minimax algorithm.
- ✓ As we have seen in the Minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree.
- ✓ Since we cannot eliminate the exponent, but we can cut it to half.
- ✓ Hence there is a technique by which without checking each node of the game tree we can compute the correct Minimax decision, and this technique is called pruning.
- ✓ This involves two threshold parameter **alpha** and **beta** for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.

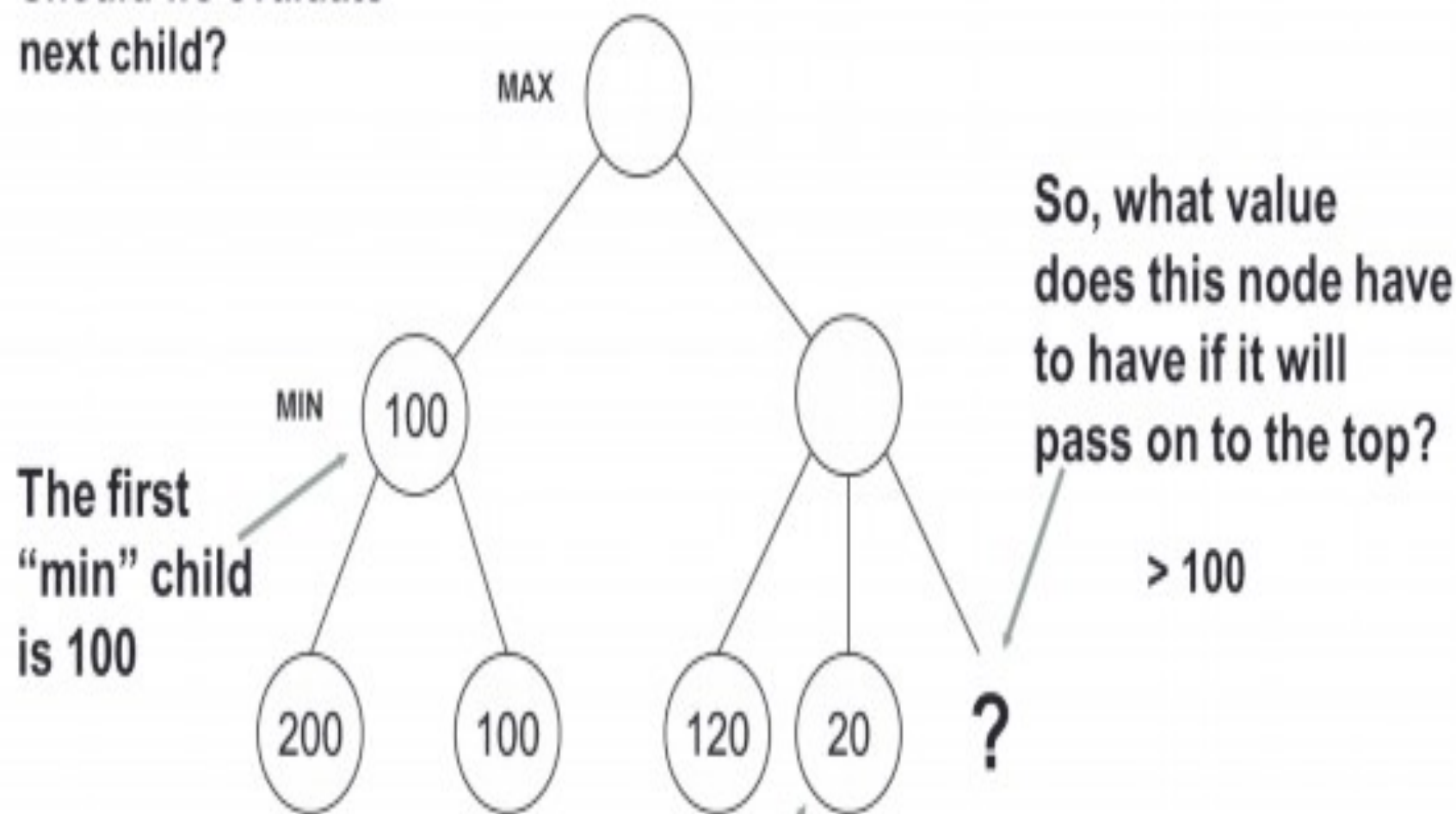


## Alpha-Beta Pruning.

- ✓ Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- ✓ The two-parameter can be defined as:
  - Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is  $-\infty$ .
  - Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is  $+\infty$ .

## Example

Should we evaluate  
next child?



The first  
"min" child  
is 100

So, what value  
does this node have  
to have if it will  
pass on to the top?

> 100

But the "min" will select the 20 instead



## Alpha-Beta Pruning.

- The **Max** player will only update the value of **alpha**.
- The **Min** player will only update the value of **beta**.
- While backtracking the tree, the **node values will be passed to upper nodes** instead of values of alpha and beta.
- We will only **pass the alpha, beta values to the child nodes**.



## Alpha-Beta Pruning.

Condition for Alpha-beta pruning:

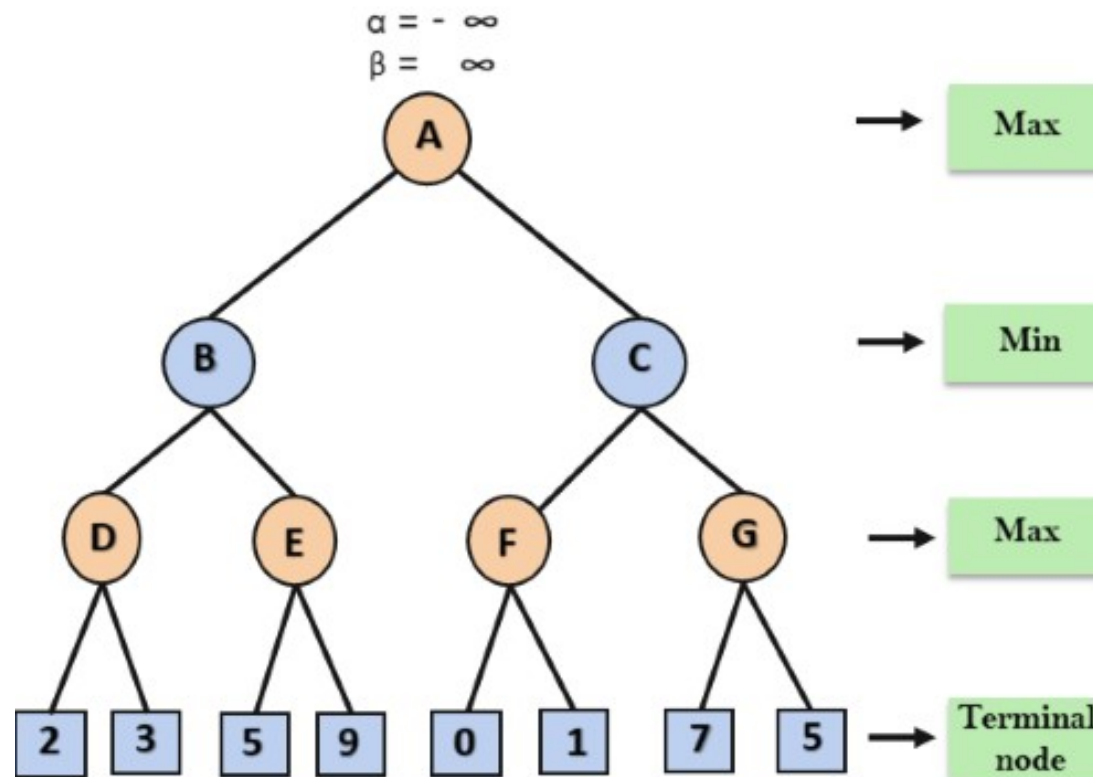
The main condition which required for alpha-beta pruning is:  $\alpha \geq \beta$





## Alpha-Beta Pruning.

Step 1: At the first step the, Max player will start first move from node A where  $\alpha = -\infty$  and  $\beta = +\infty$ , these value of alpha and beta passed down to node B where again  $\alpha = -\infty$  and  $\beta = +\infty$ , and Node B passes the same value to its child D.





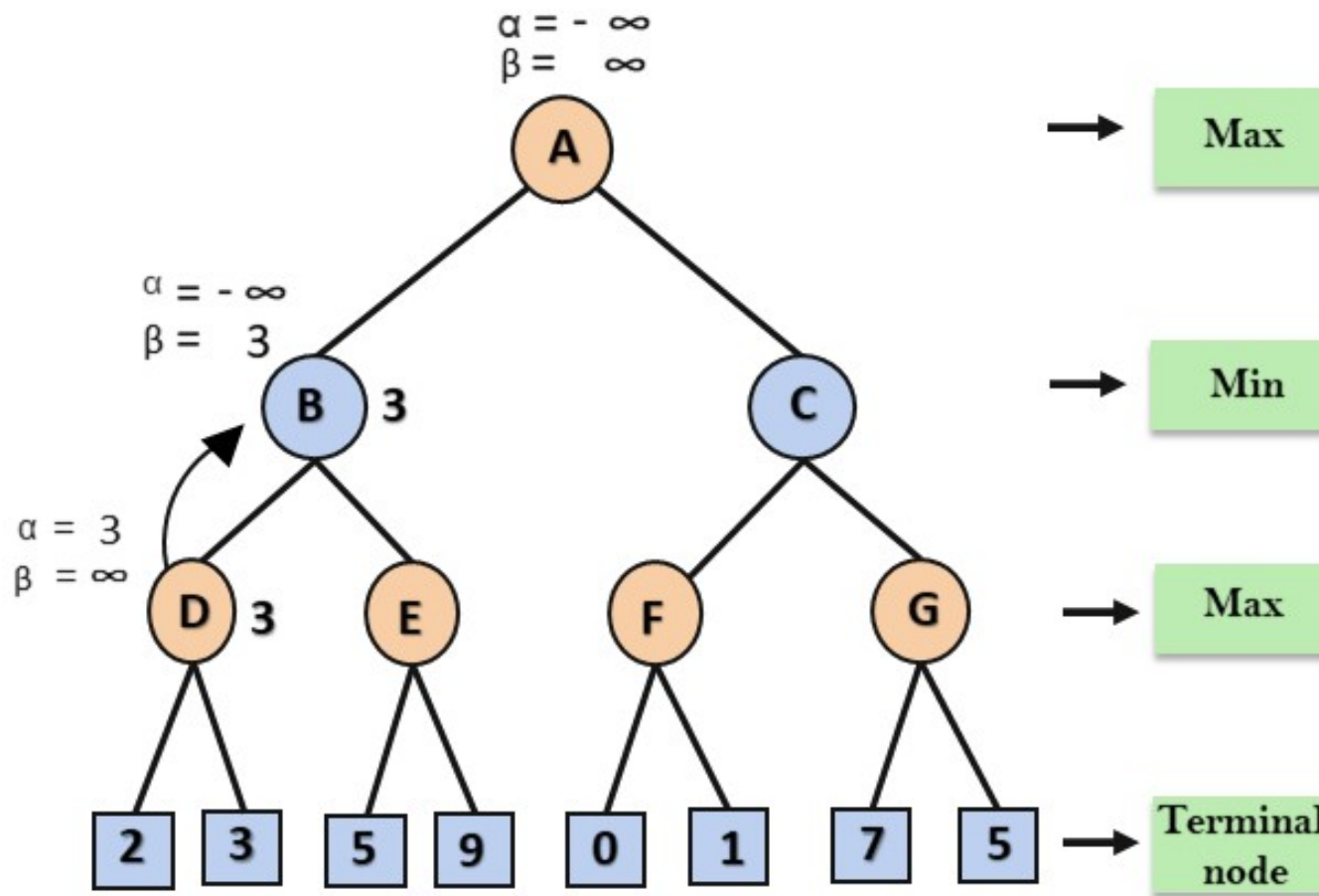
## Alpha-Beta Pruning.

Step 2: At Node D, the value of  $\alpha$  will be calculated as its turn for Max. The value of  $\alpha$  is compared with firstly 2 and then 3, and the  $\max(2, 3) = 3$  will be the value of  $\alpha$  at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of  $\beta$  will change as this is a turn of Min, Now  $\beta = +\infty$ , will compare with the available subsequent nodes value, i.e.  $\min(\infty, 3) = 3$ , hence at node B now  $\alpha = -\infty$ , and  $\beta = 3$ .



## Alpha-Beta Pruning.



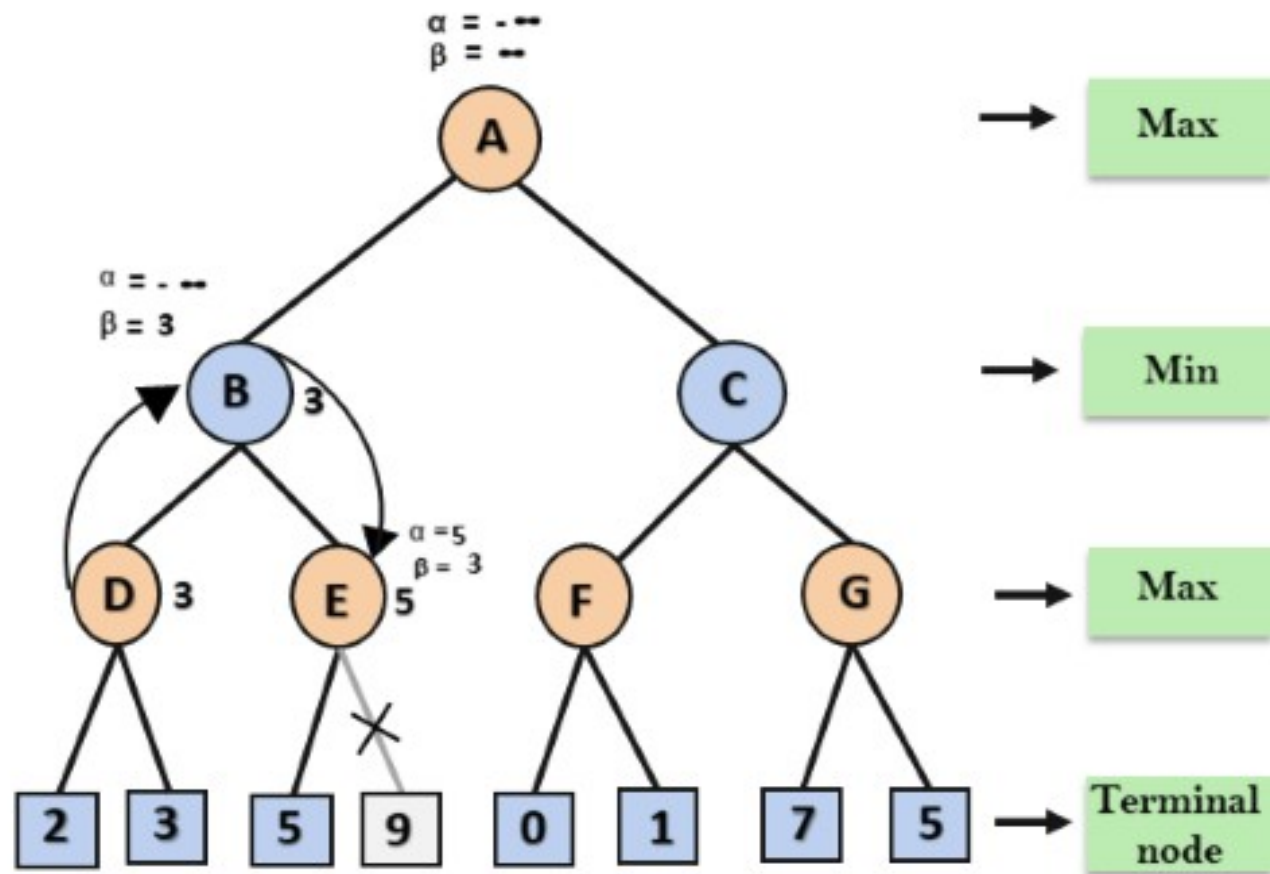
In the next step, algorithm traverse the next successor of Node B which is node E, and the values of  $\alpha = -\infty$ , and  $\beta = 3$  will also be passed.



## Alpha-Beta Pruning.

In the next step, algorithm traverse the next successor of Node B which is node E, and the values of  $\alpha = -\infty$ , and  $\beta = 3$  will also be passed.

Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so  $\max(-\infty, 5) = 5$ , hence at node E  $\alpha = 5$  and  $\beta = 3$ , where  $\alpha \geq \beta$ , so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.







## Alpha-Beta Pruning.

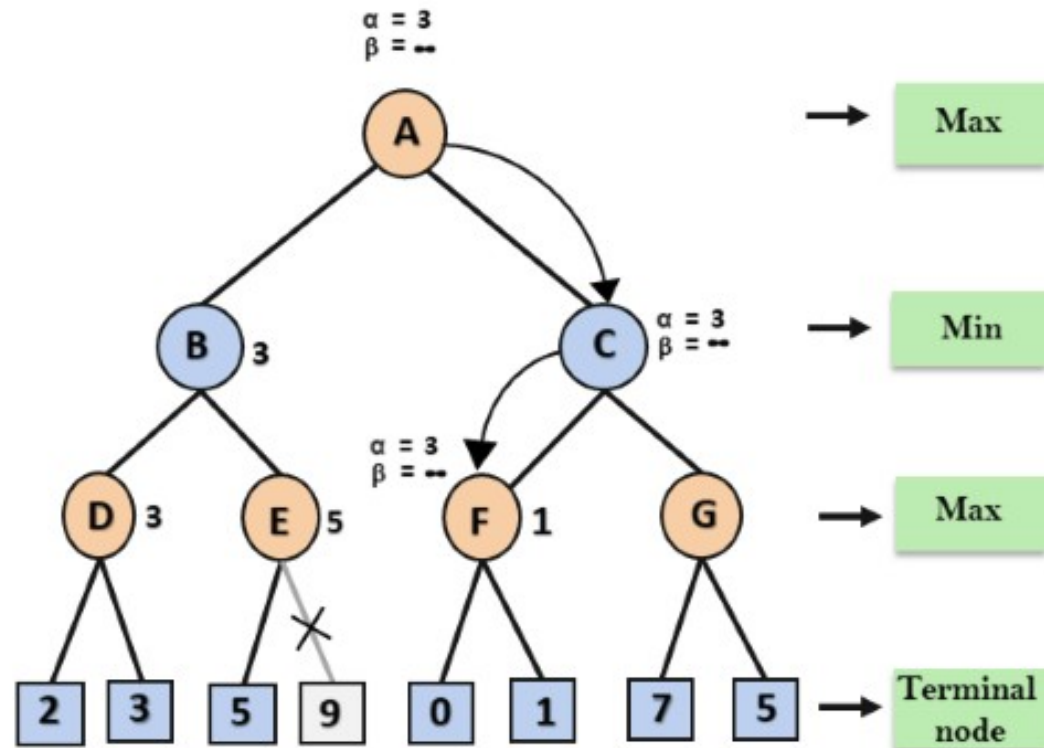
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as  $\max(-\infty, 3) = 3$ , and  $\beta = +\infty$ , these two values now passes to right successor of A which is Node C.

At node C,  $\alpha = 3$  and  $\beta = +\infty$ , and the same values will be passed on to node F.

Step 6: At node F, again the value of  $\alpha$  will be compared with left child which is 0, and  $\max(3, 0) = 3$ , and then compared with right child which is 1, and  $\max(3, 1) = 3$  still  $\alpha$  remains 3, but the node value of F will become 1.



# Alpha-Beta Pruning.



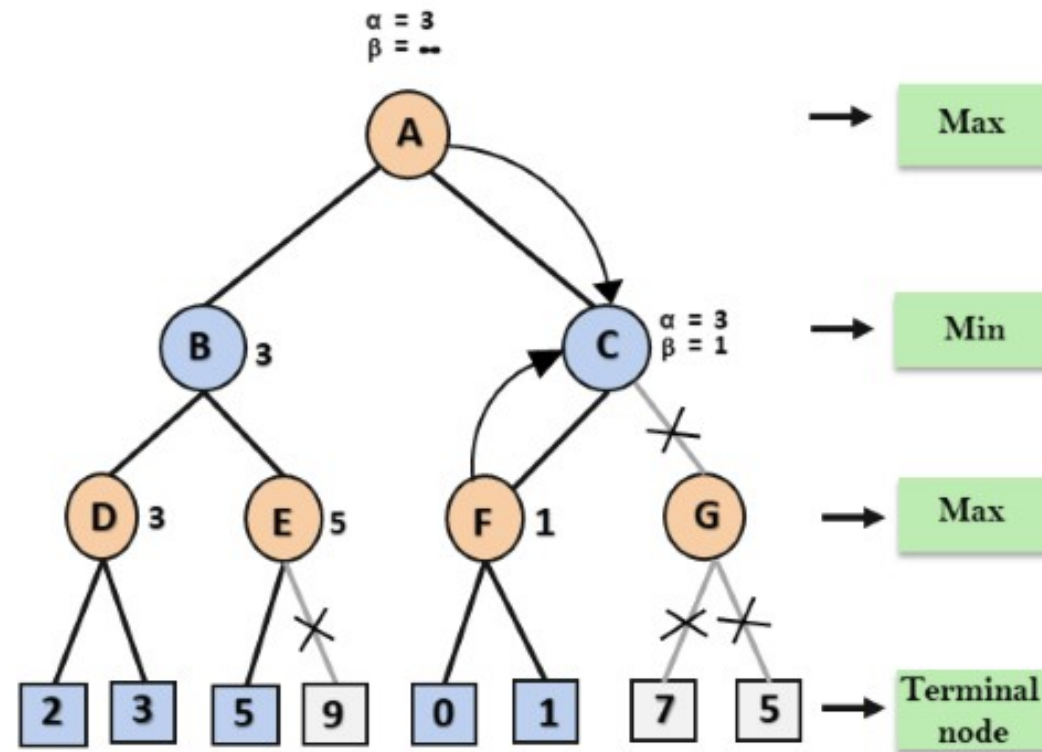


## Alpha-Beta Pruning.

Step 7: Node F returns the node value 1 to node C, at C  $\alpha = 3$  and  $\beta = +\infty$ , here the value of beta will be changed, it will compare with 1 so  $\min(\infty, 1) = 1$ . Now at C,  $\alpha = 3$  and  $\beta = 1$ , and again it satisfies the condition  $\alpha \geq \beta$ , so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



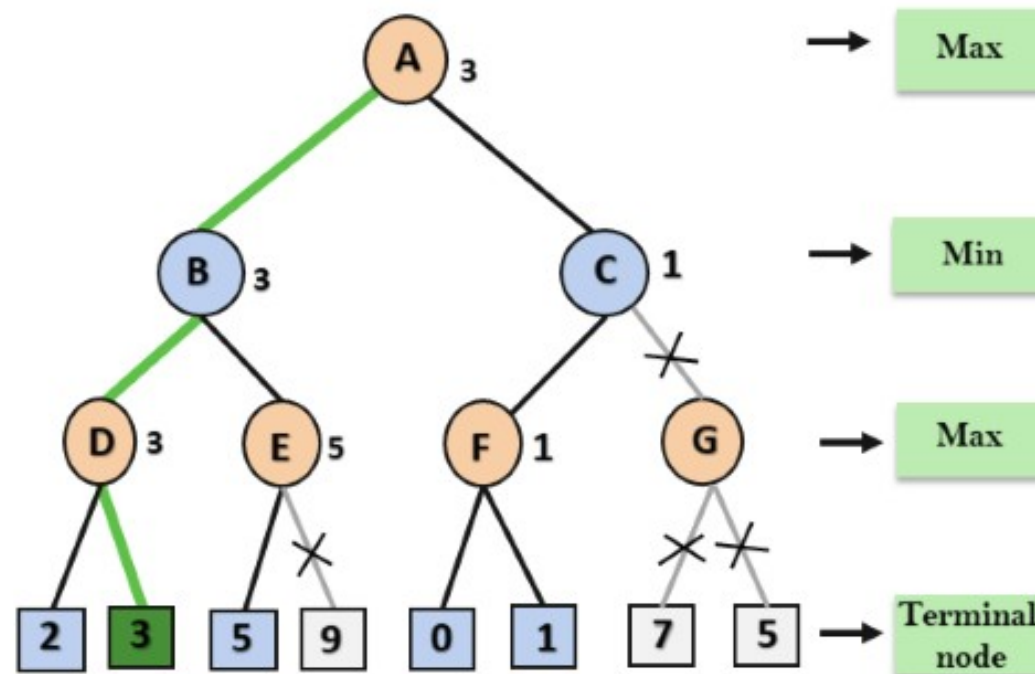
# Alpha-Beta Pruning.






## Alpha-Beta Pruning.

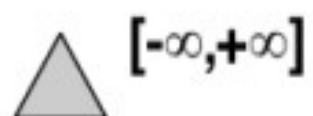
Step 8: C now returns the value of 1 to A here the best value for A is  $\max(3, 1) = 3$ . Following is the final game tree which is showing the nodes which are computed and nodes which has never computed. Hence the **optimal value for the maximizer is 3** for this example.





# $\alpha$ - $\beta$ pruning: example 1

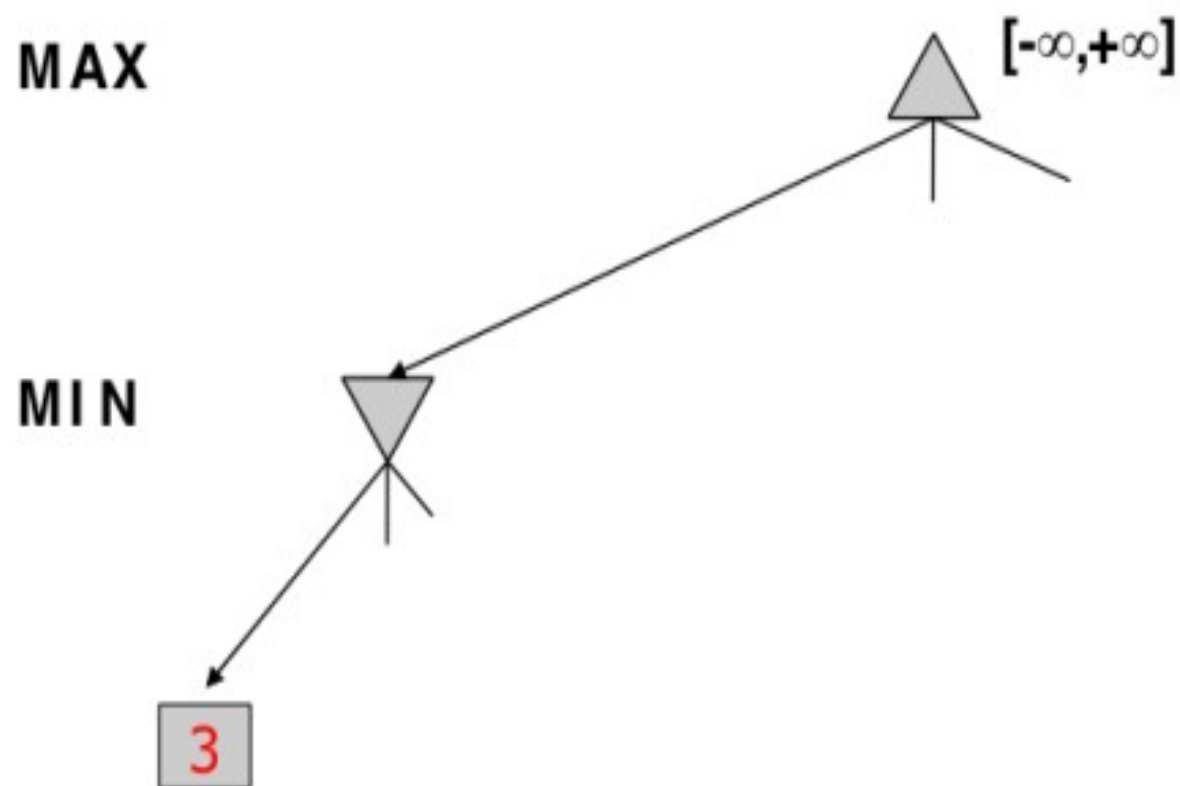
MAX



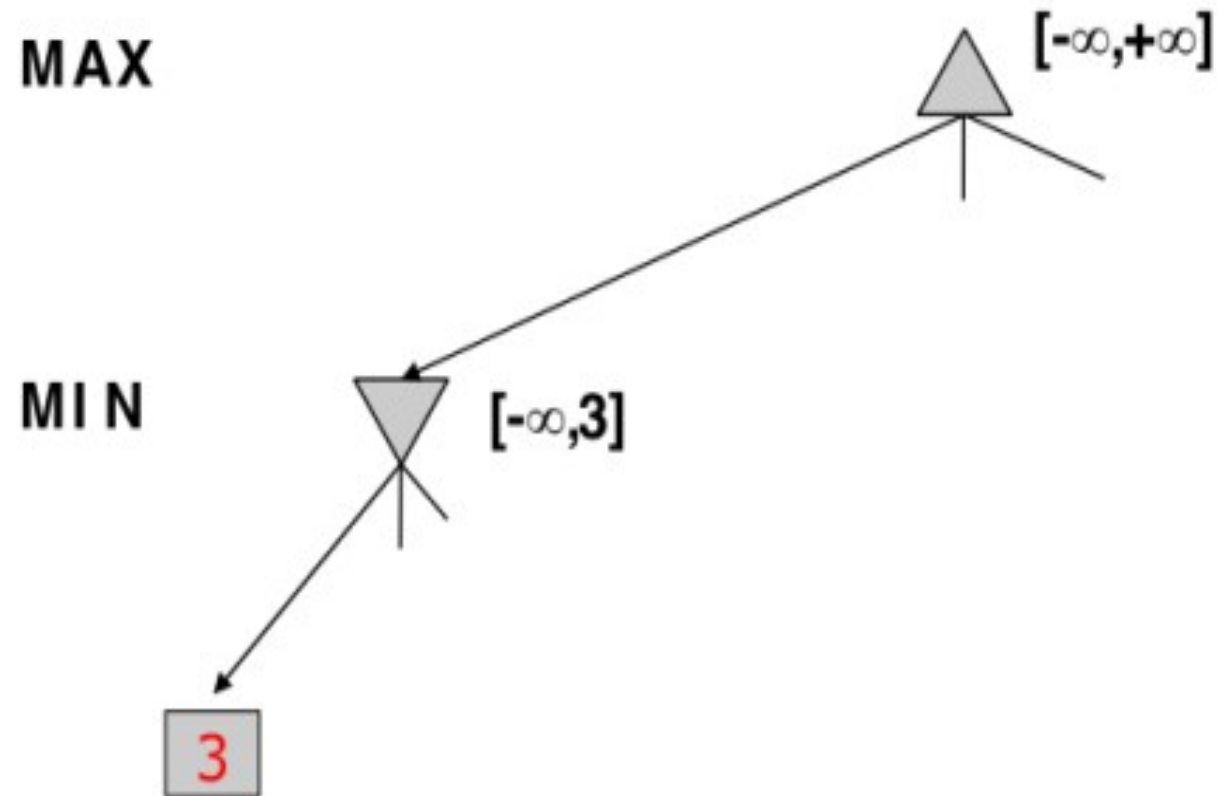
MIN



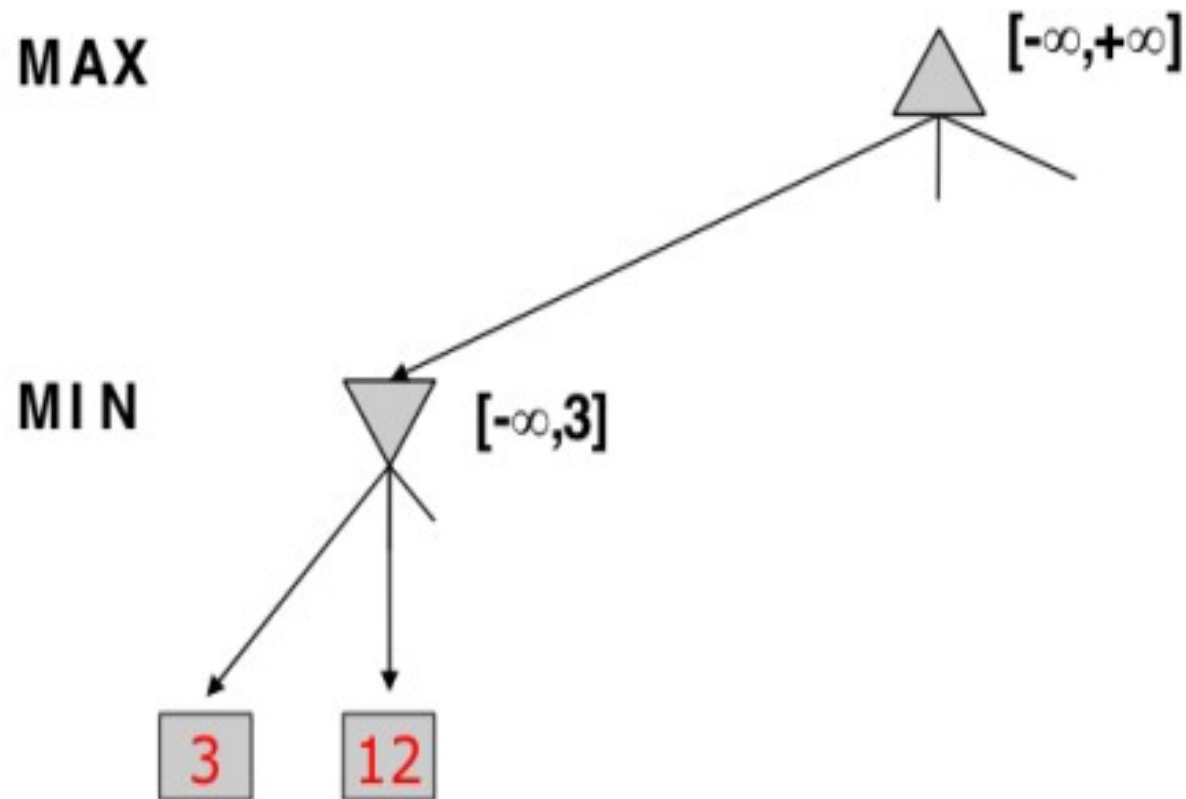
# $\alpha$ - $\beta$ pruning: example 1



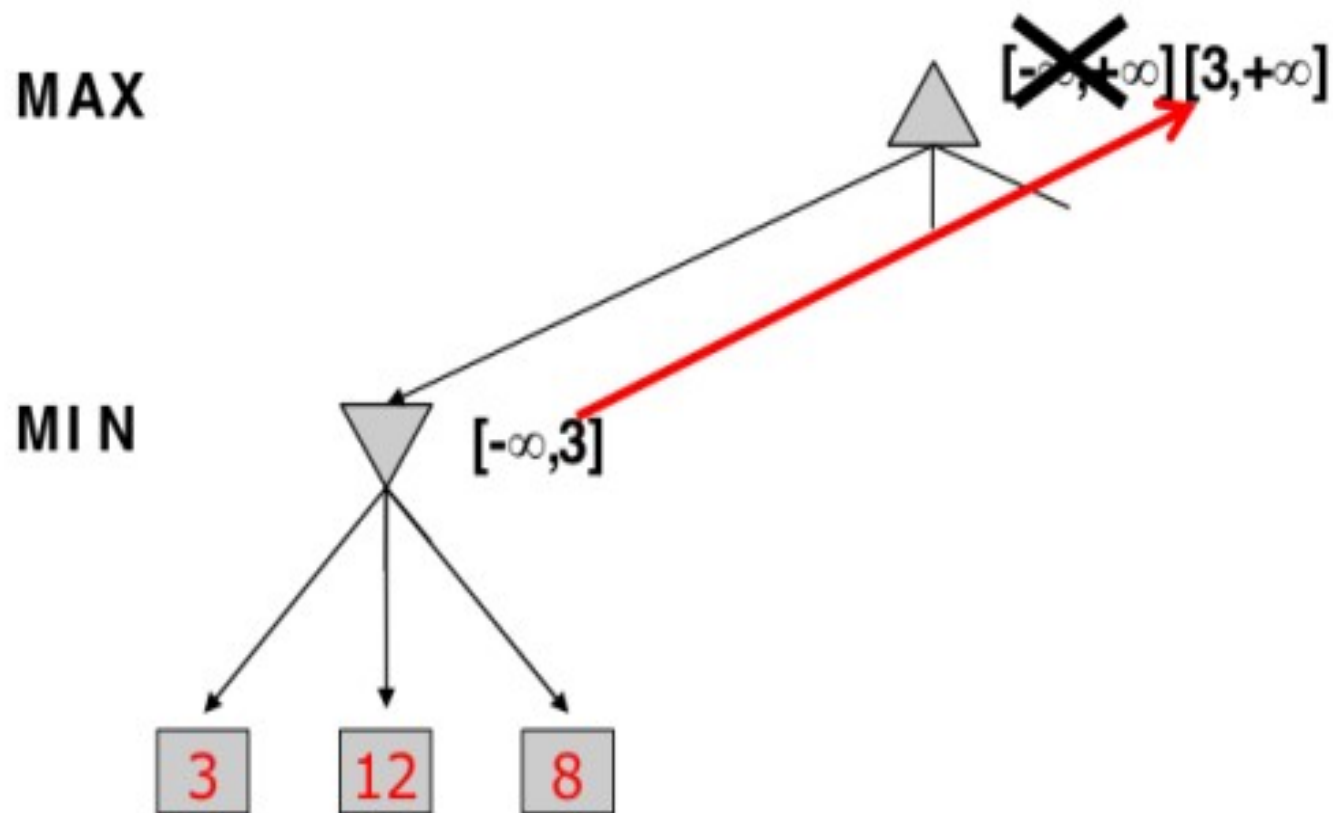
# $\alpha$ - $\beta$ pruning: example 1



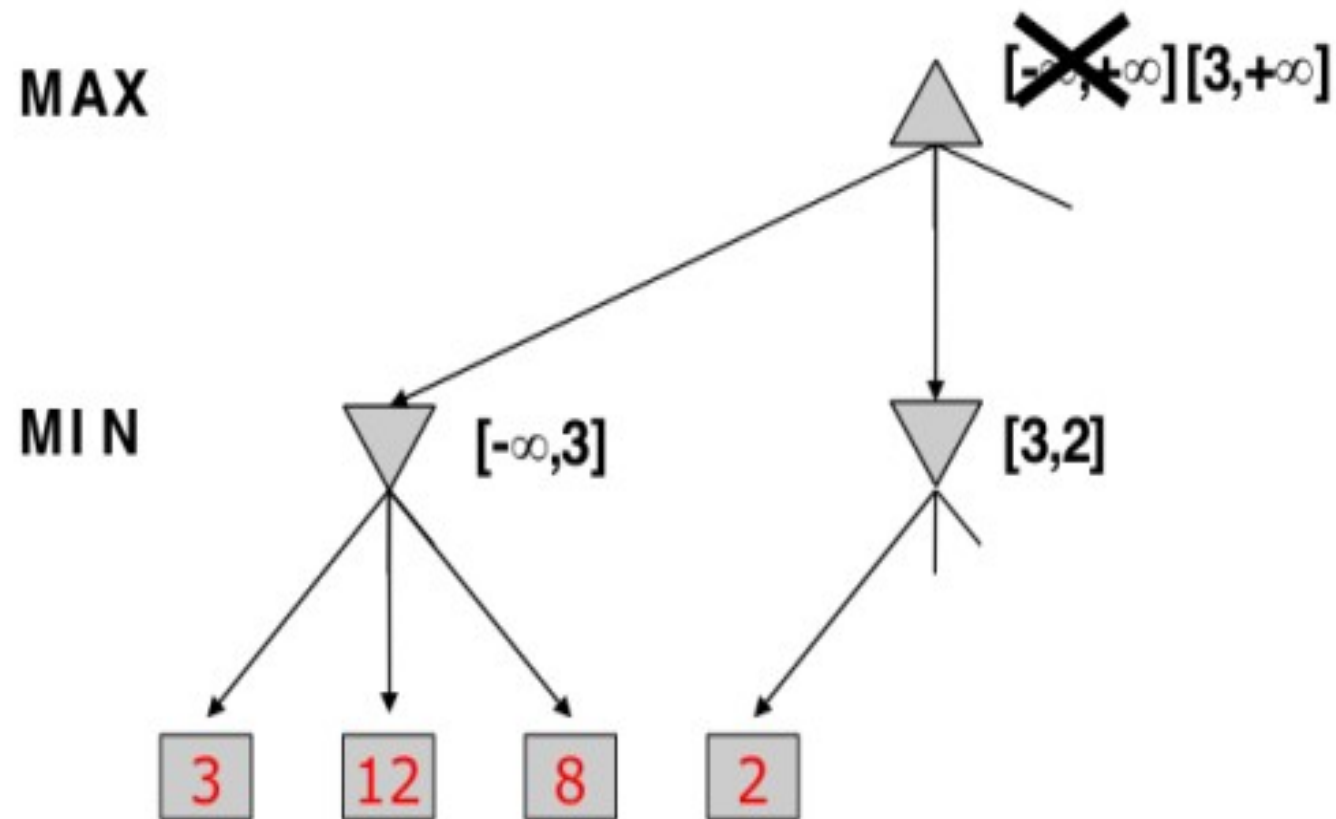
# $\alpha$ - $\beta$ pruning: example 1



# $\alpha$ - $\beta$ pruning: example 1



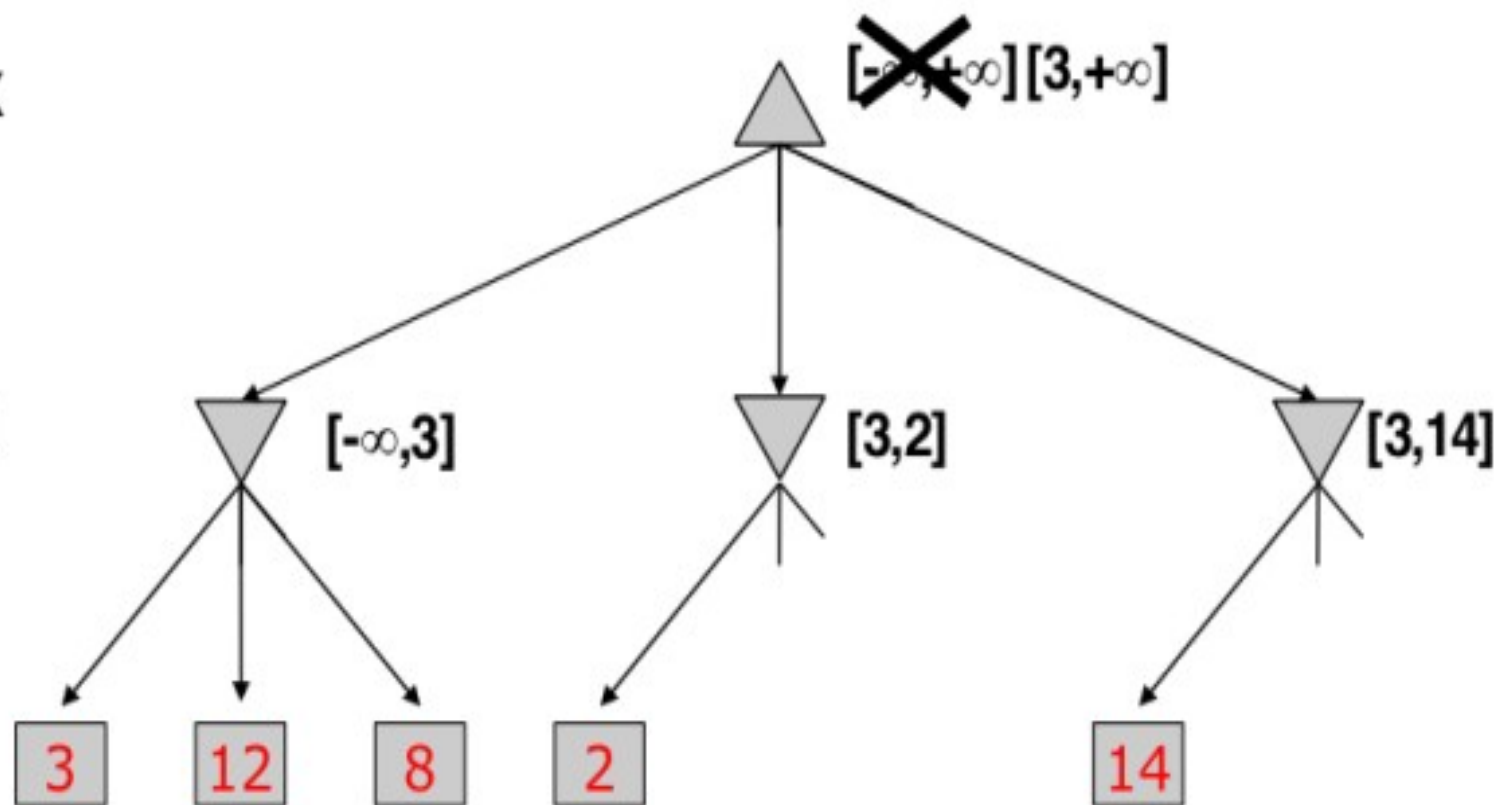
# $\alpha$ - $\beta$ pruning: example 1



# $\alpha$ - $\beta$ pruning: example 1

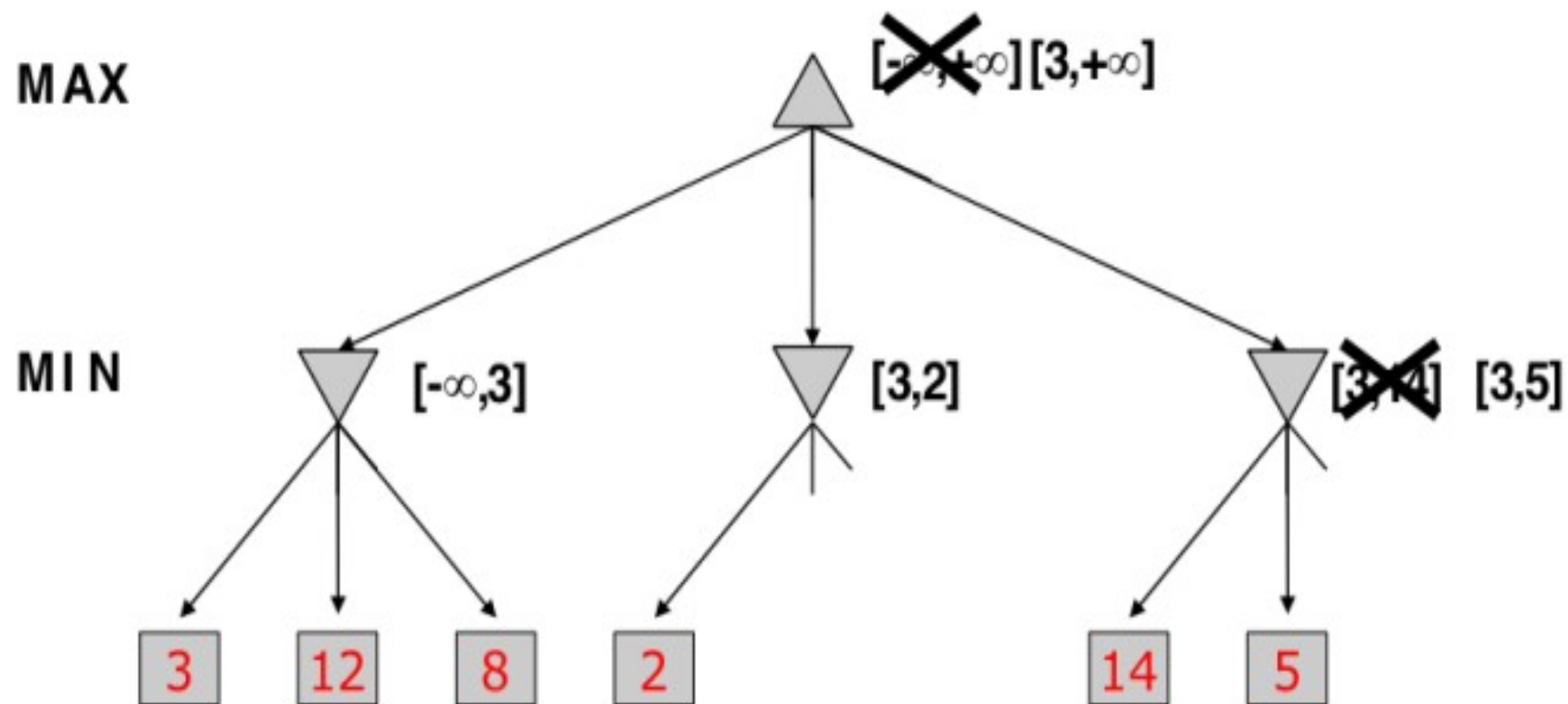
MAX

MIN





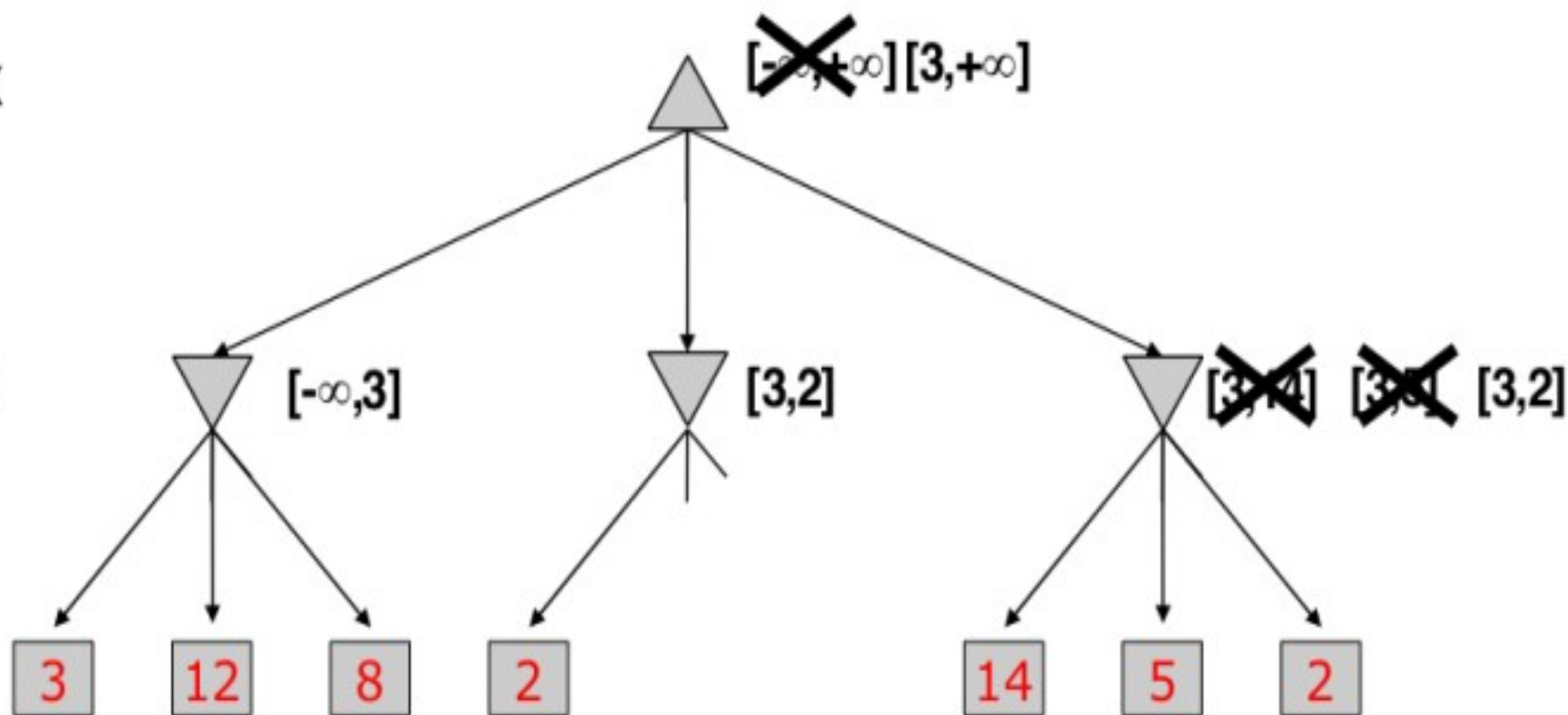
# $\alpha$ - $\beta$ pruning: example 1



# $\alpha$ - $\beta$ pruning: example 1

MAX

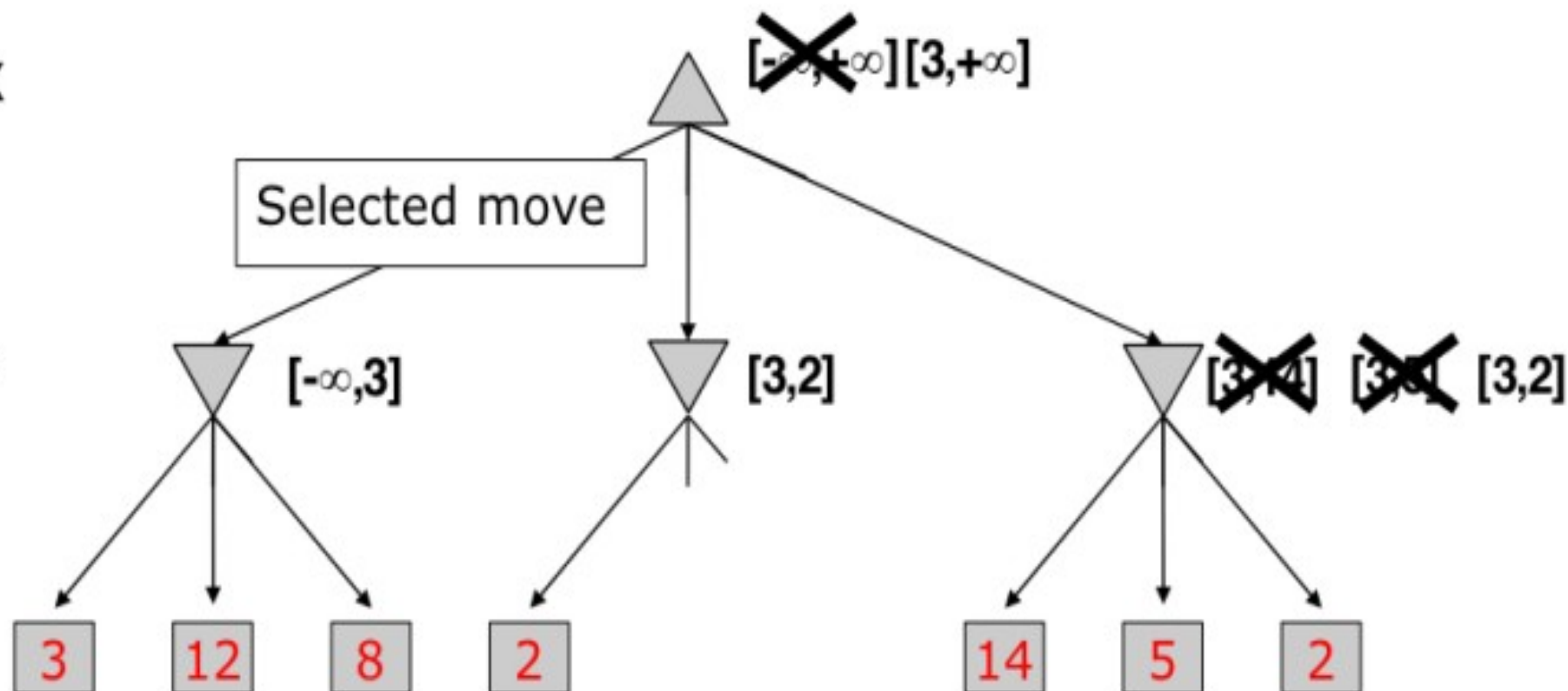
MIN



# $\alpha$ - $\beta$ pruning: example 1

MAX

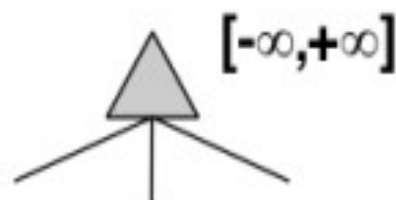
MIN



$$\begin{aligned}
 \text{MINIMAX-VALUE}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\
 &= \max(3, \min(2, x, y), 2) \\
 &= \max(3, z, 2) \quad \text{where } z \leq 2 \\
 &= 3.
 \end{aligned}$$

# $\alpha$ - $\beta$ pruning: example 2

MAX

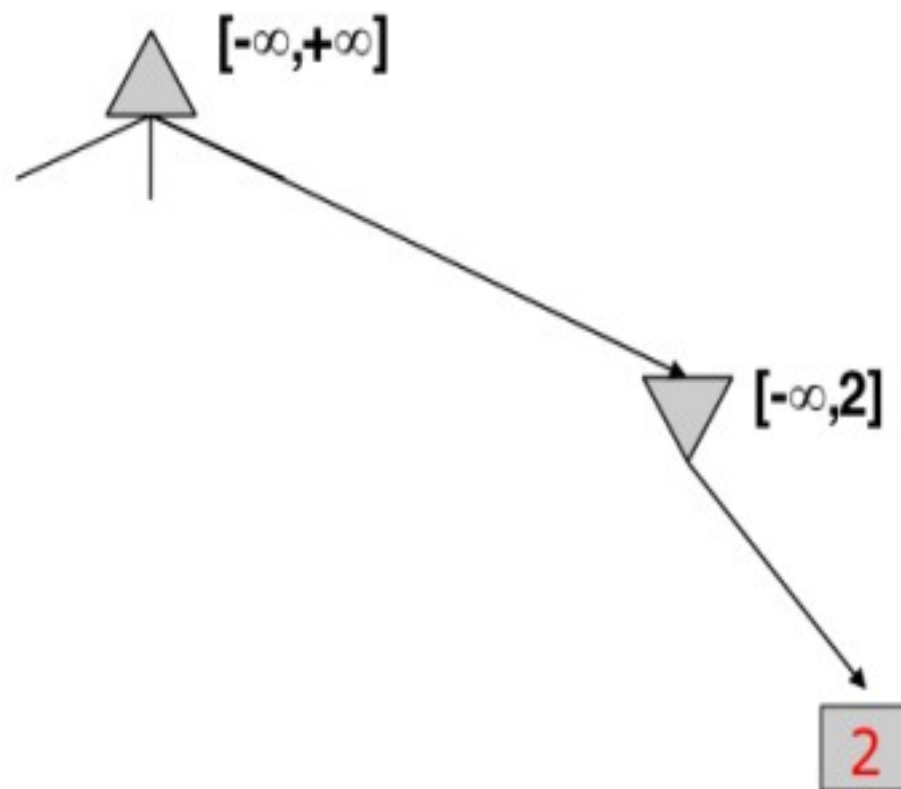


MIN

# $\alpha$ - $\beta$ pruning: example 2

MAX

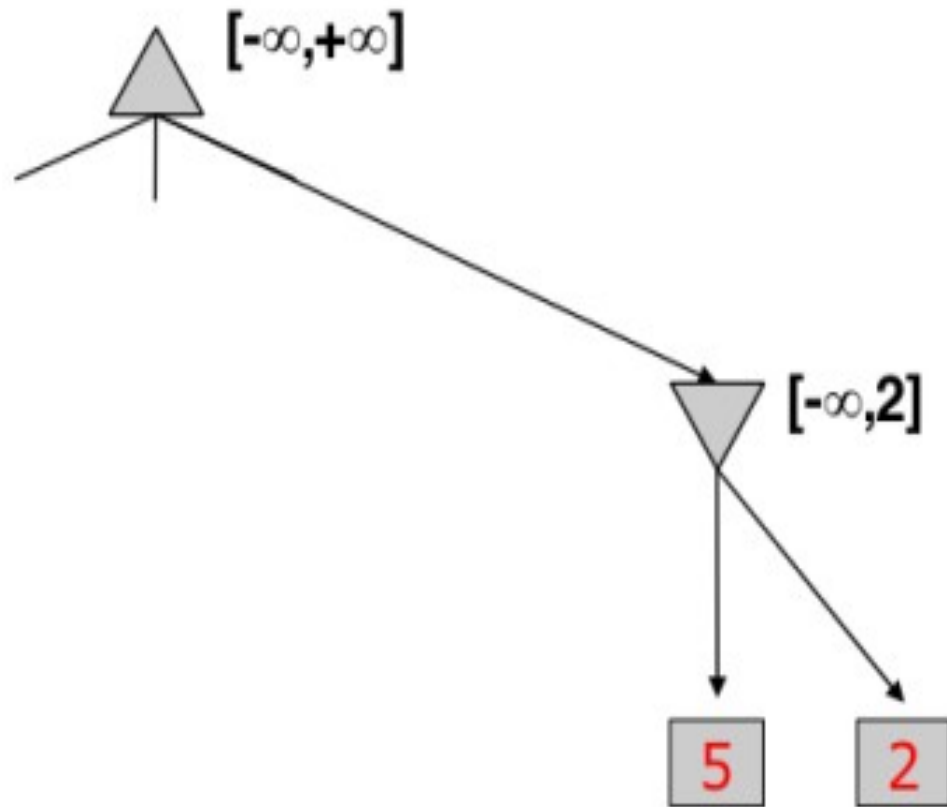
MIN



# $\alpha$ - $\beta$ pruning: example 2

MAX

MIN

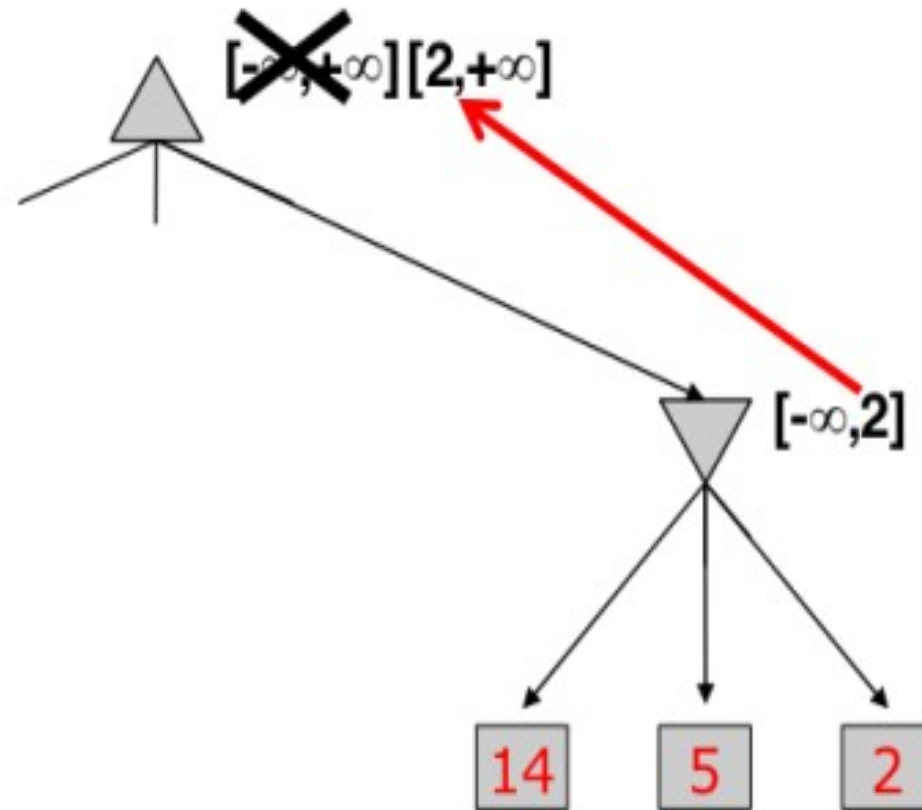




# $\alpha$ - $\beta$ pruning: example 2

MAX

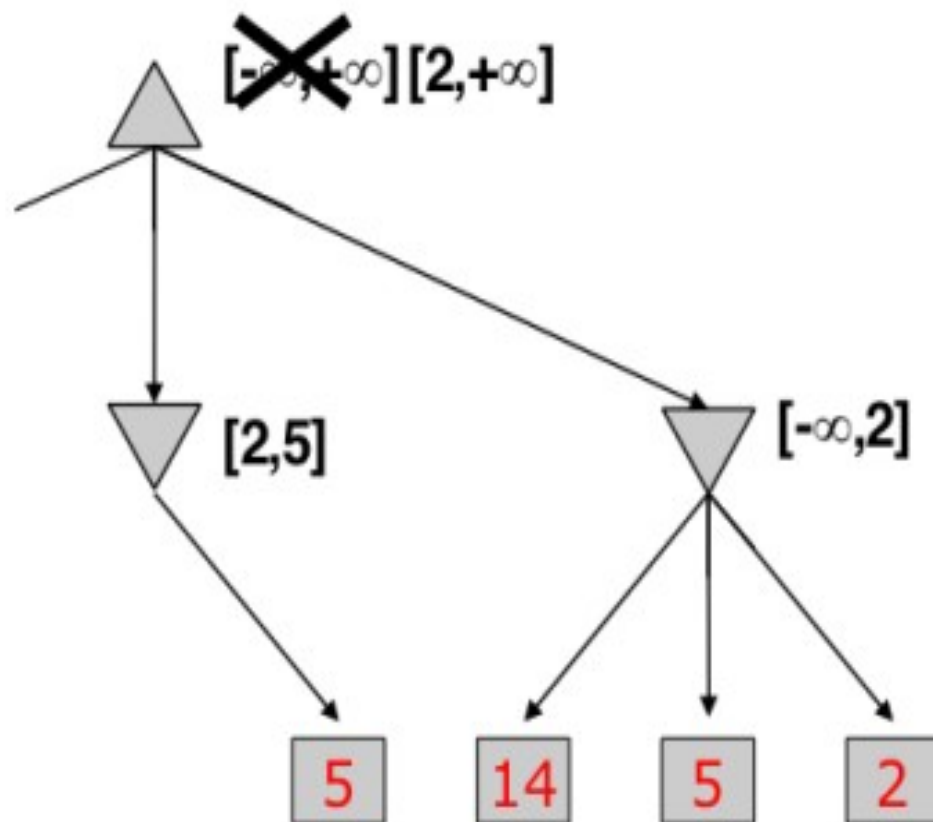
MIN



# $\alpha$ - $\beta$ pruning: example 2

MAX

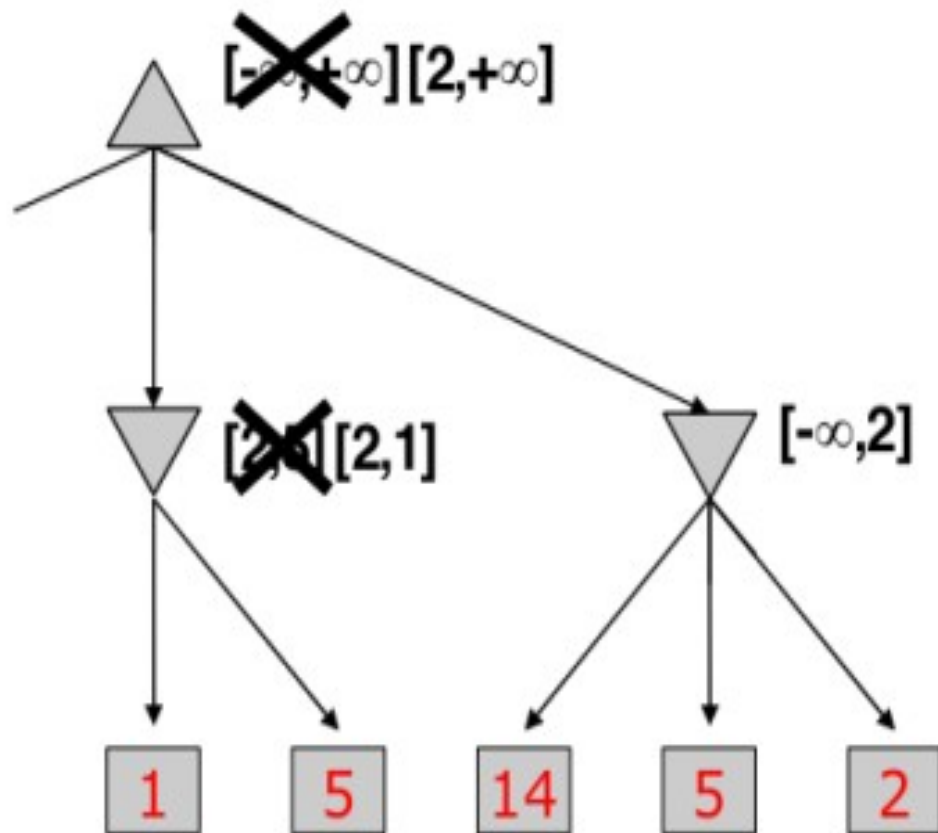
MIN



# $\alpha$ - $\beta$ pruning: example 2

MAX

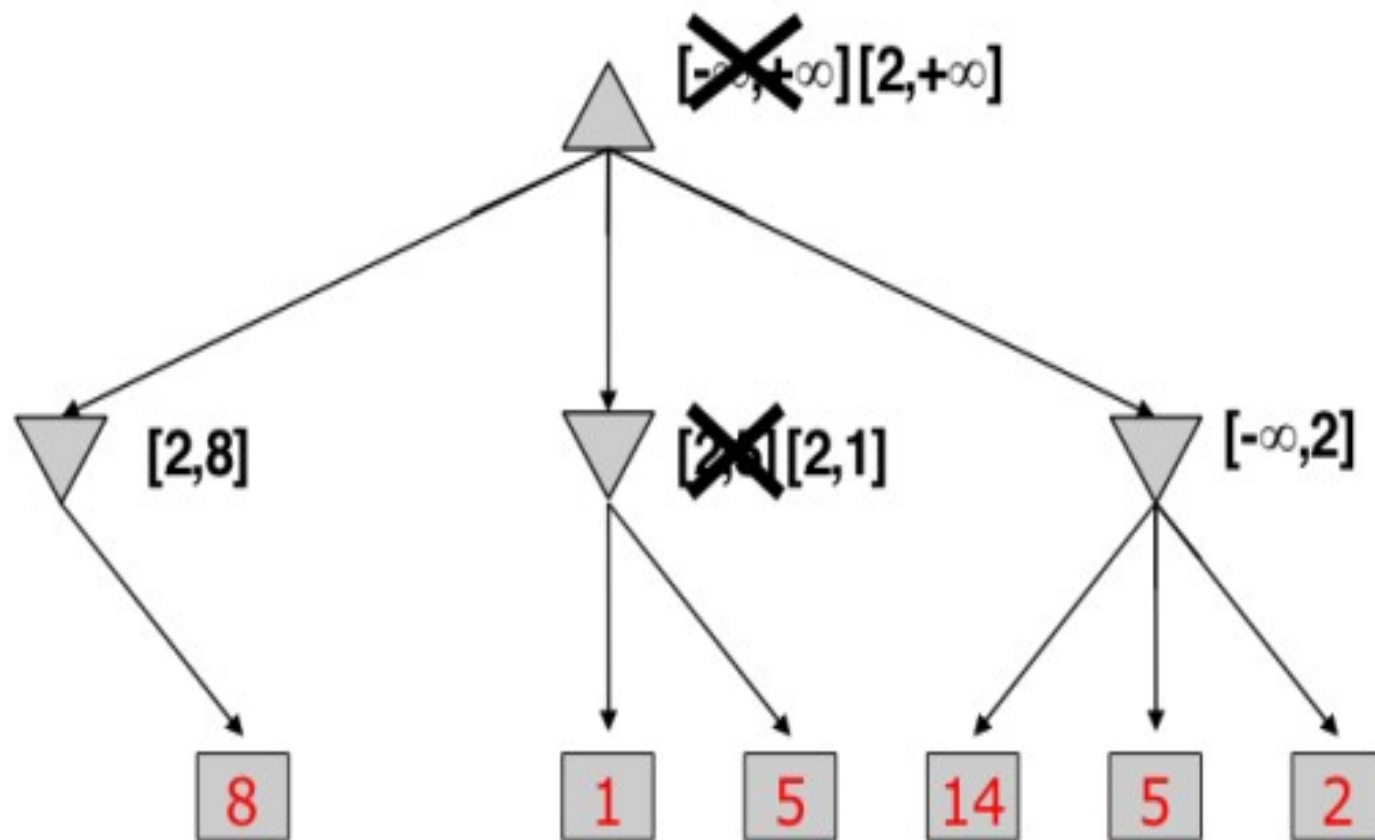
MIN



# $\alpha$ - $\beta$ pruning: example 2

MAX

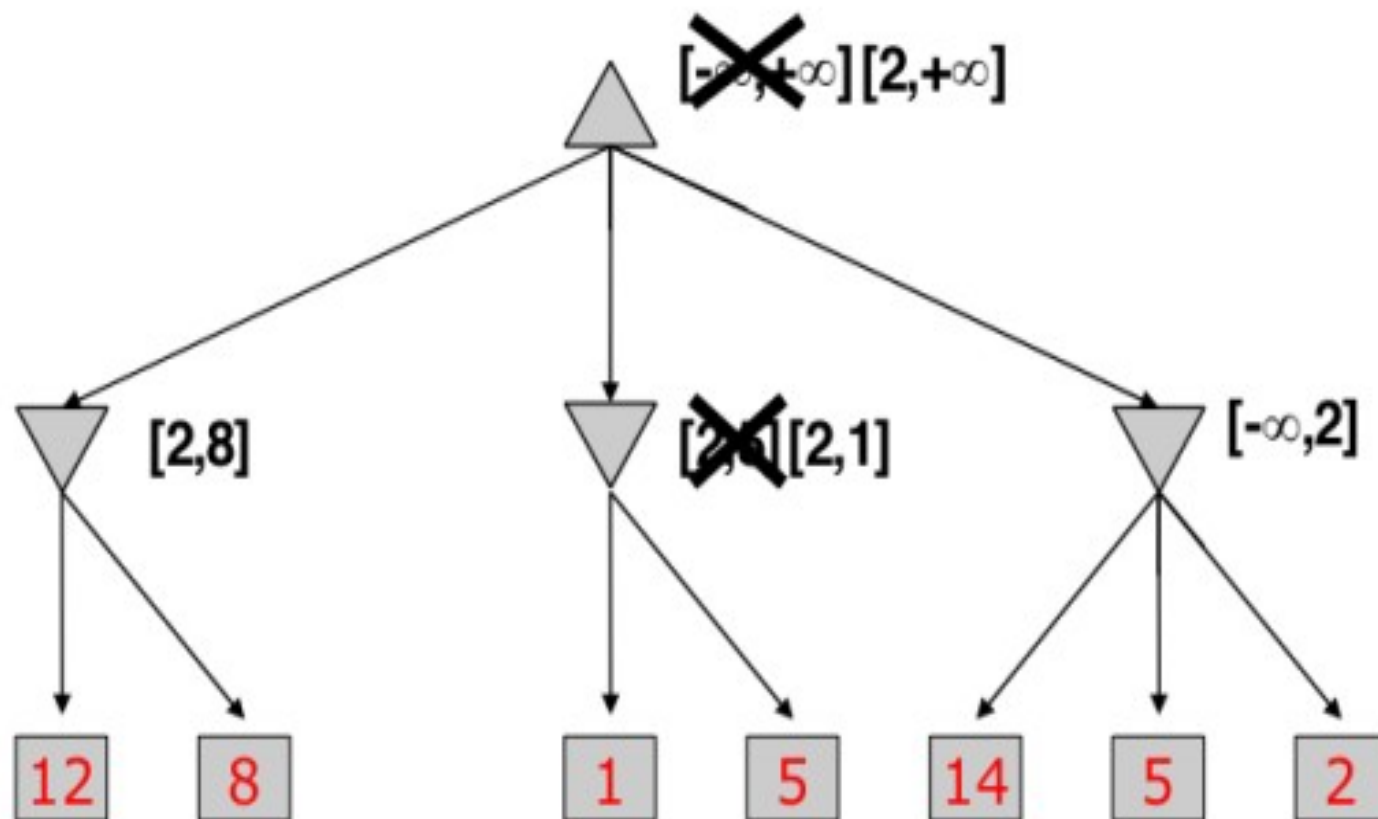
MIN



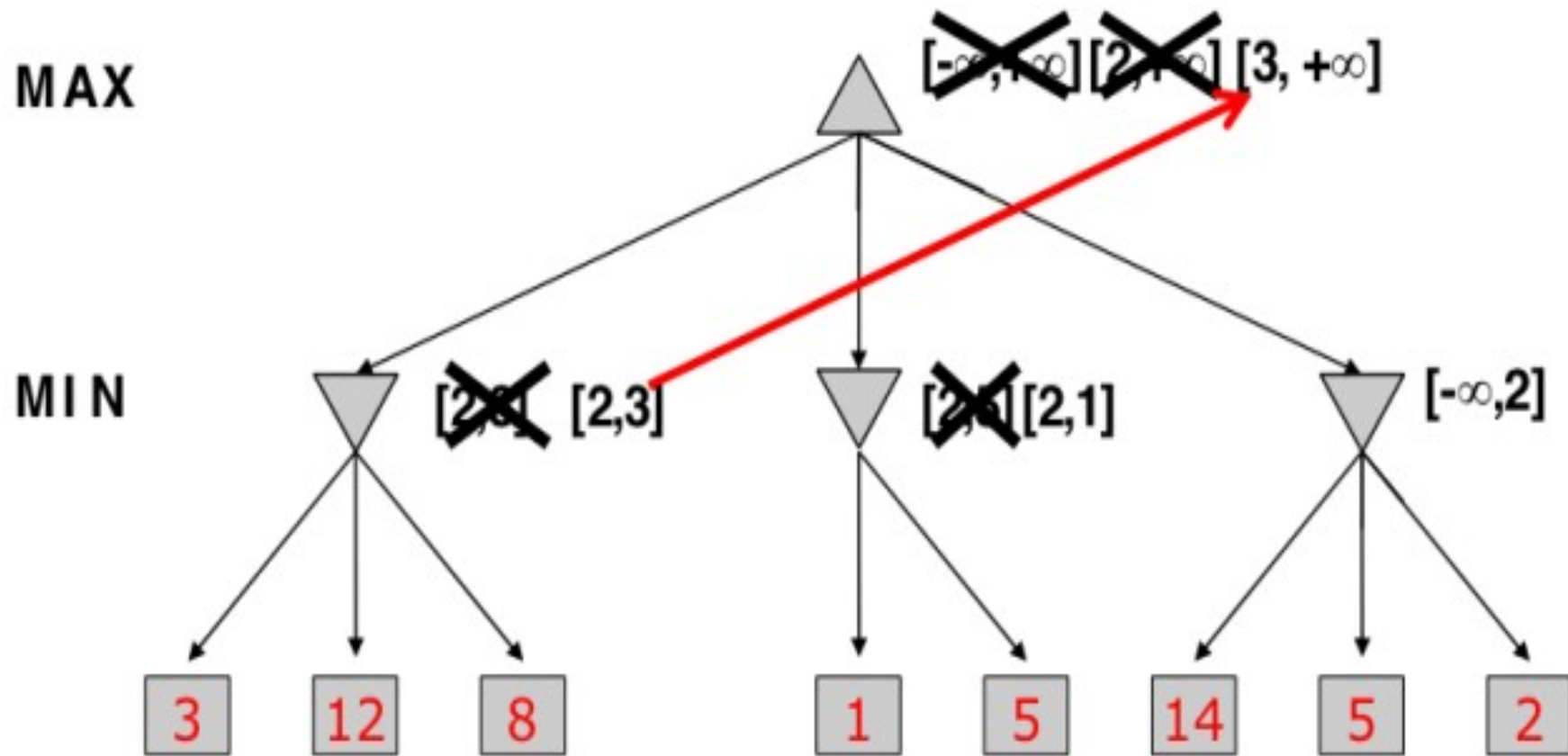
# $\alpha$ - $\beta$ pruning: example 2

MAX

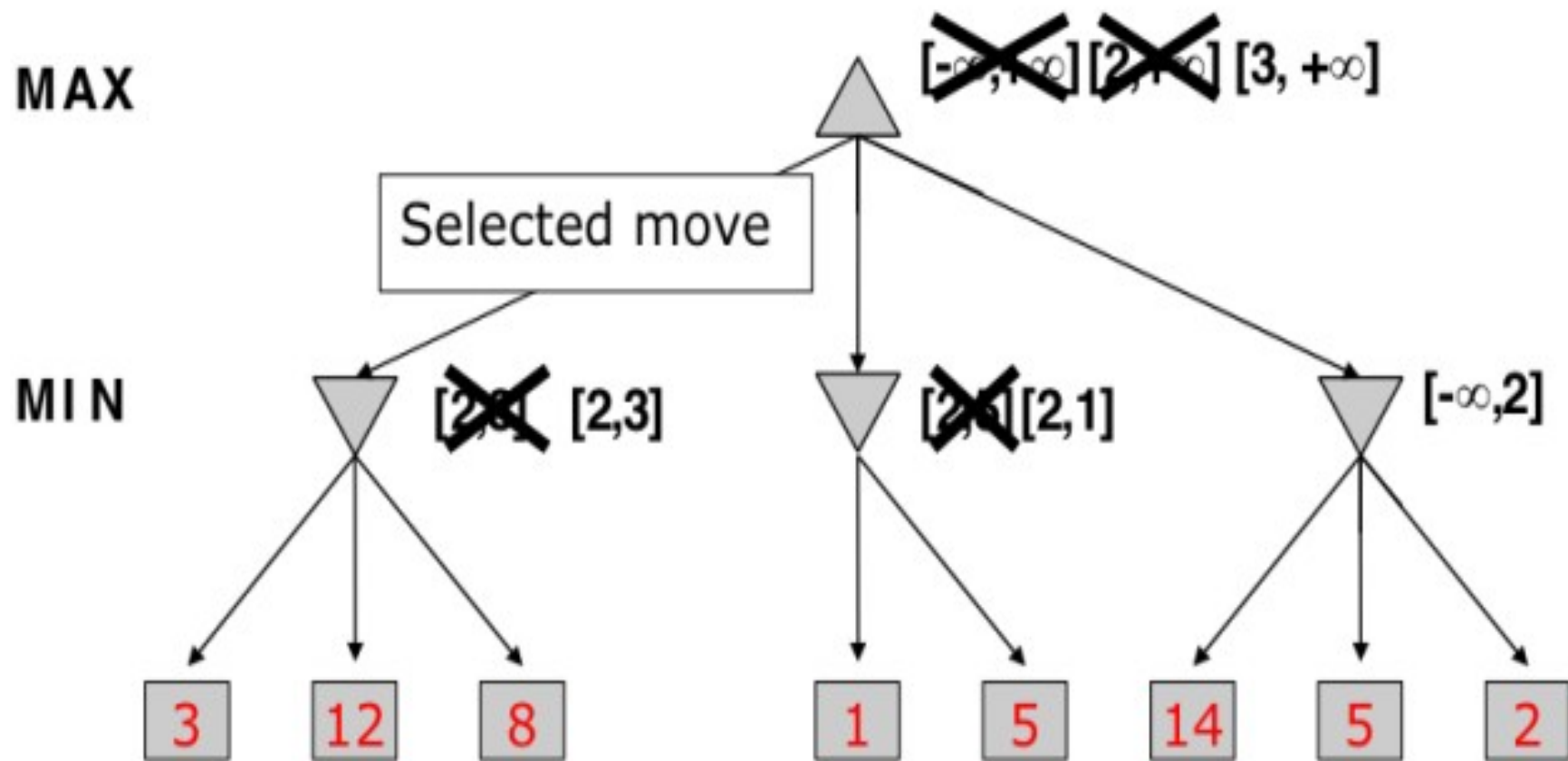
MIN



# $\alpha$ - $\beta$ pruning: example 2



# $\alpha$ - $\beta$ pruning: example 2



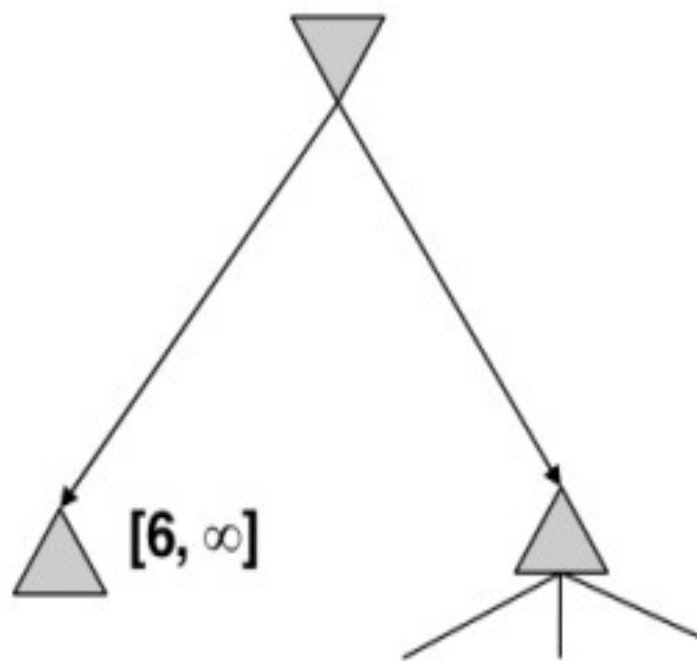


# $\alpha$ - $\beta$ pruning: example 3

MIN

MAX

MIN

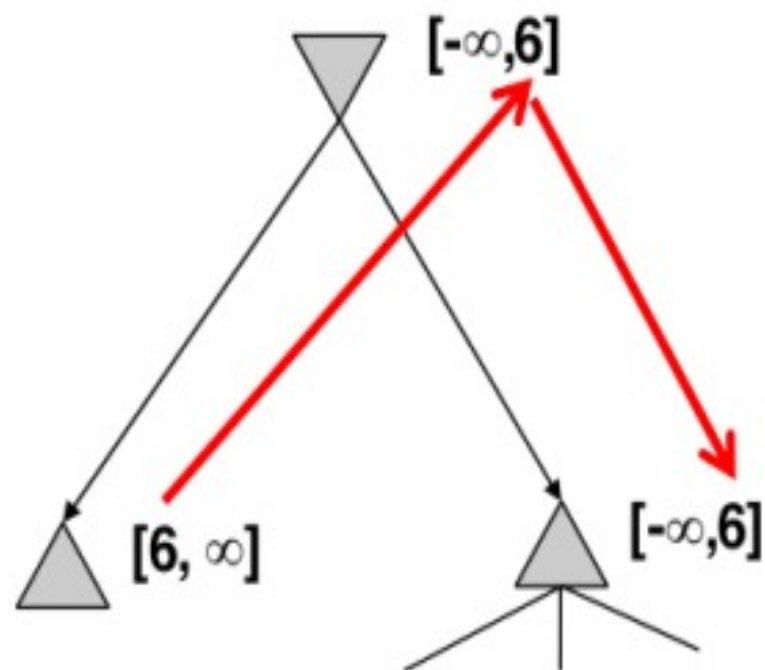


# $\alpha$ - $\beta$ pruning: example 3

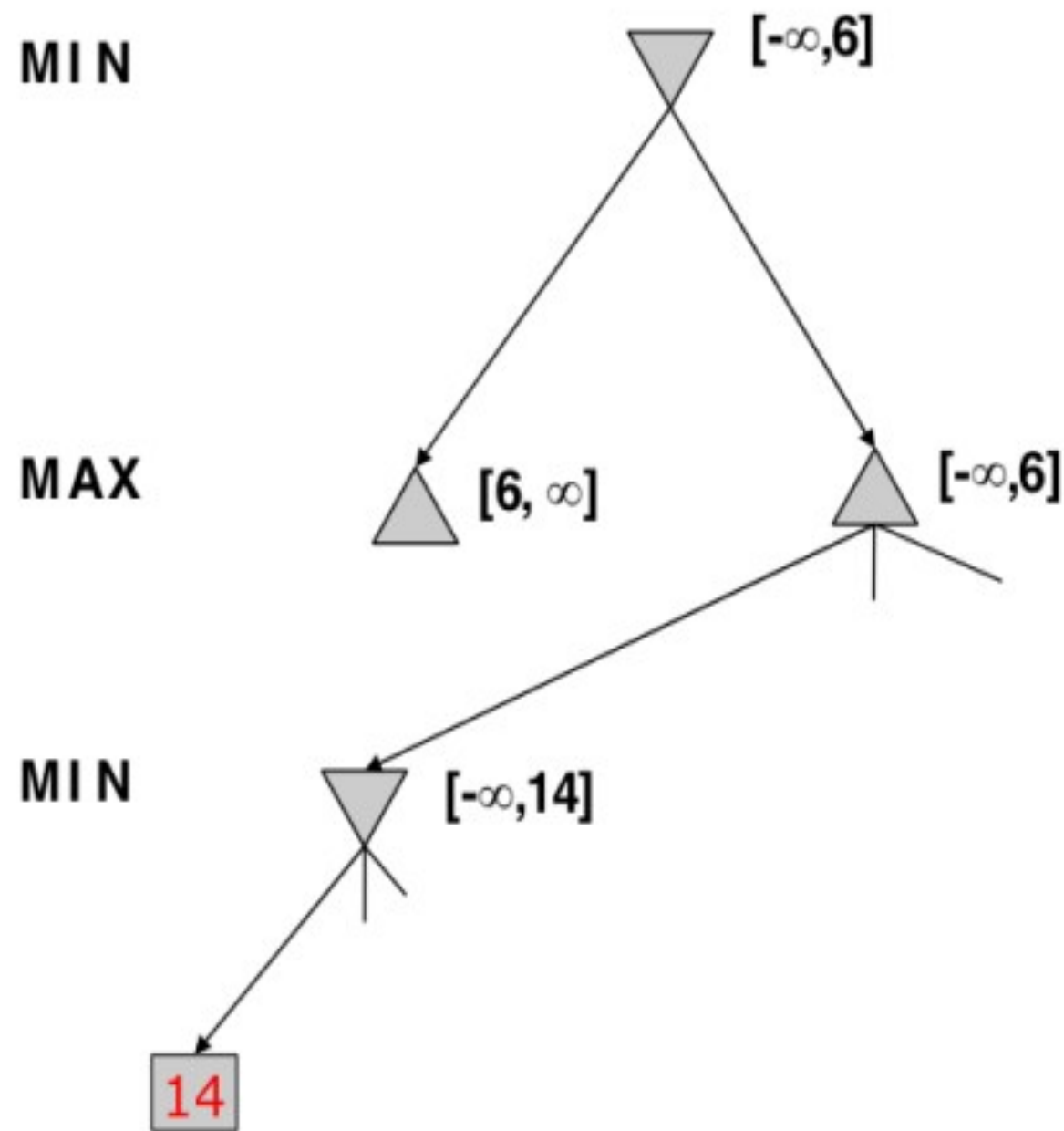
MIN

MAX

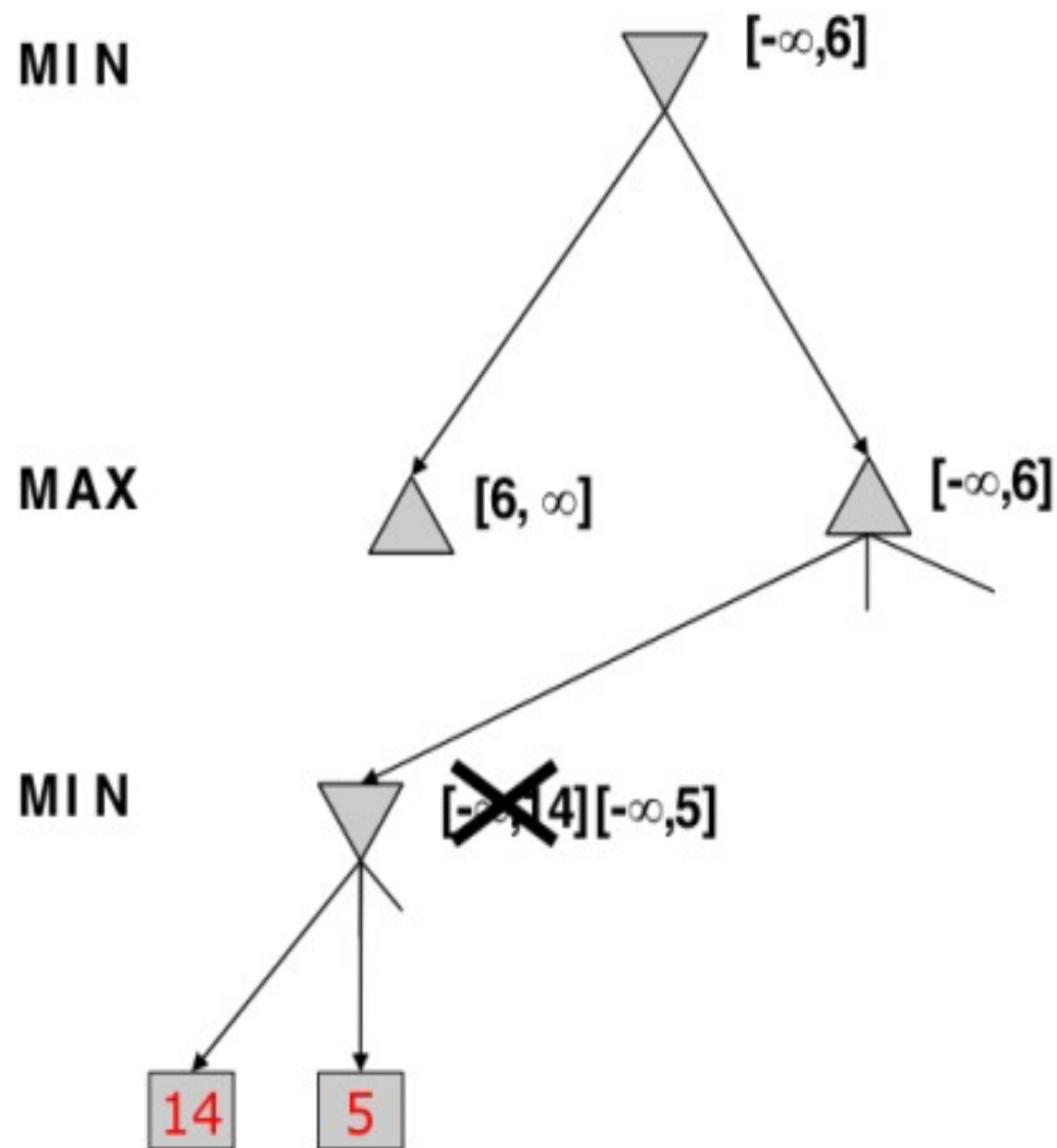
MIN



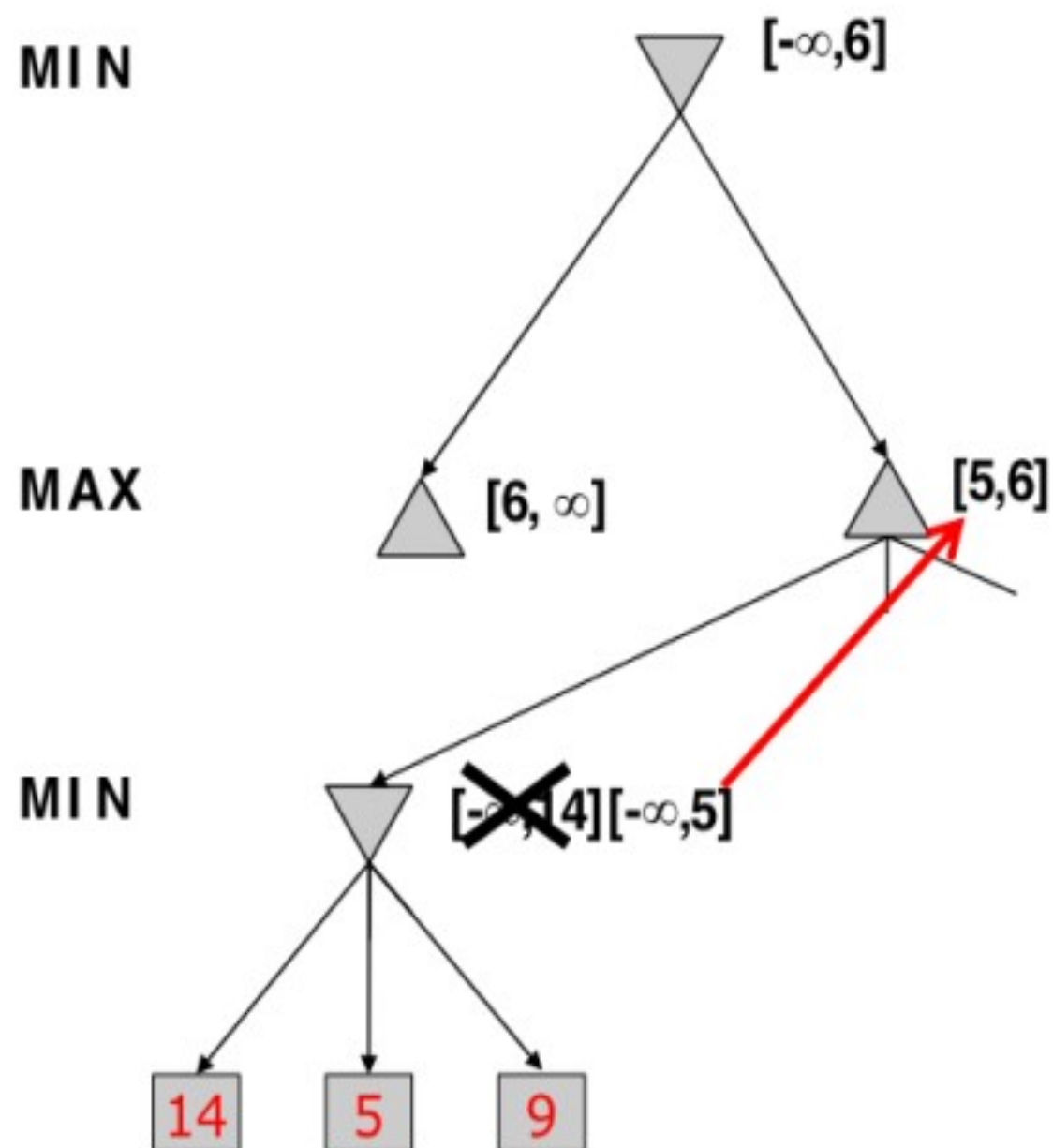
# $\alpha$ - $\beta$ pruning: example 3



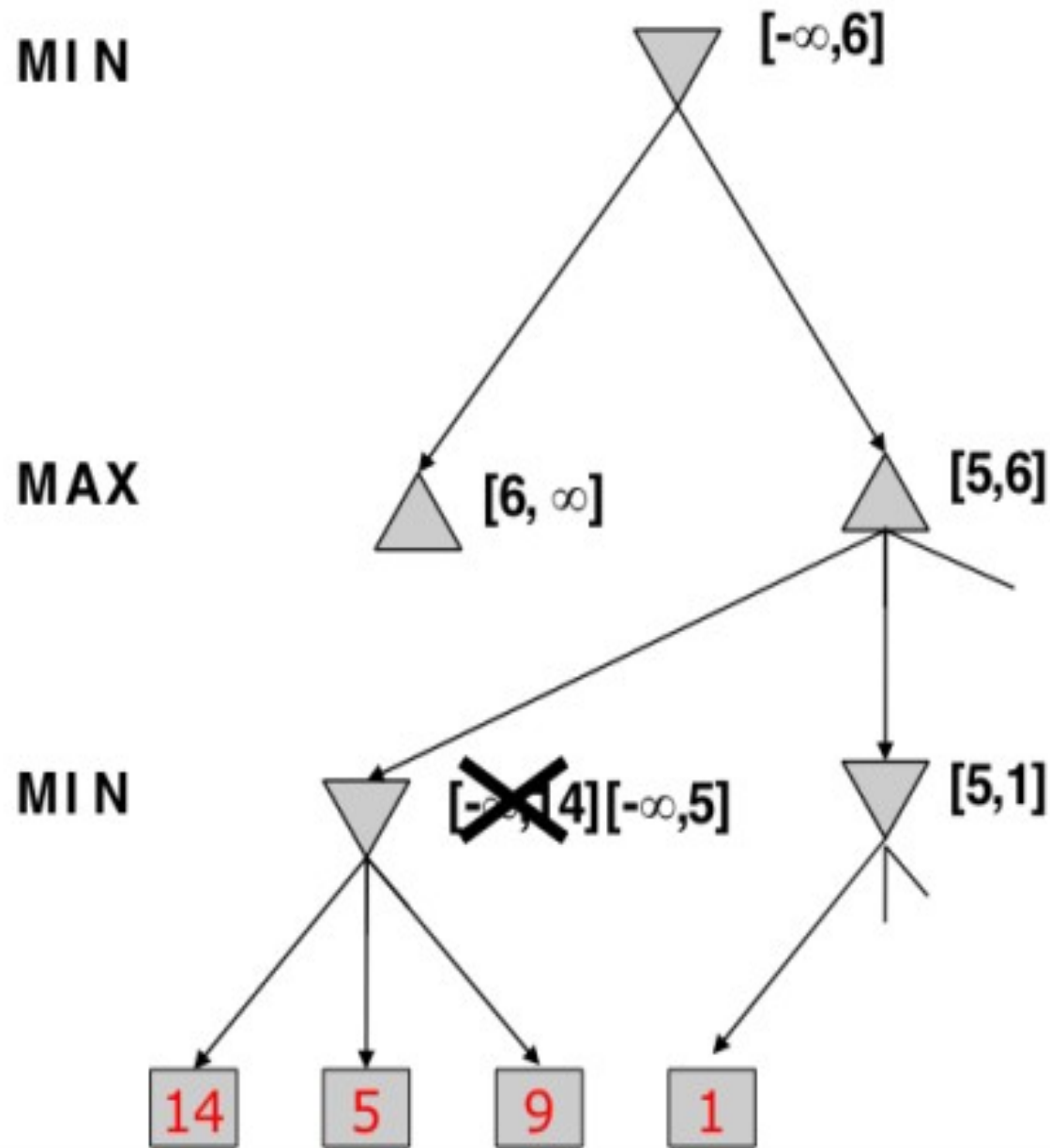
# $\alpha$ - $\beta$ pruning: example 3



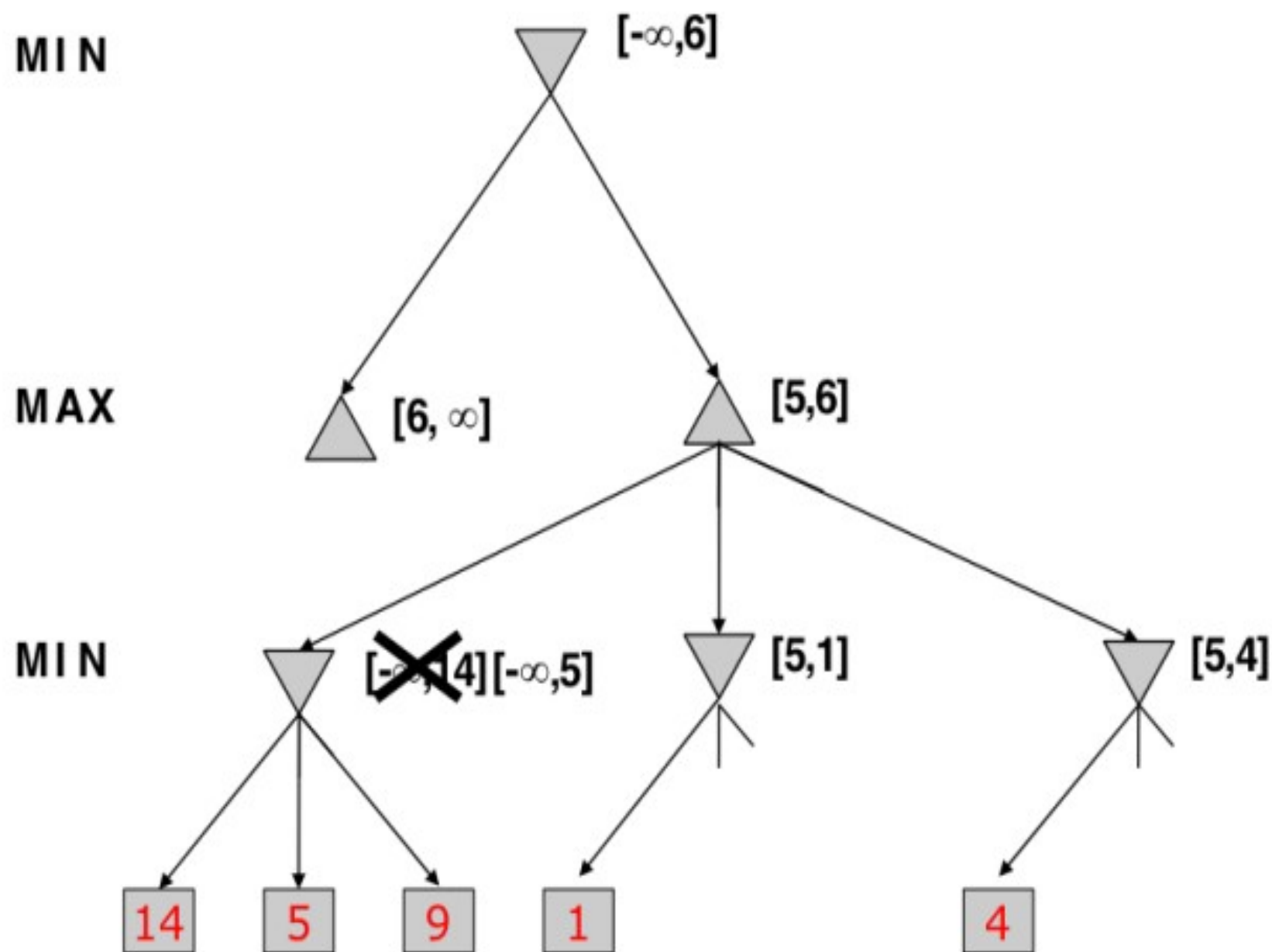
# $\alpha$ - $\beta$ pruning: example 3



# $\alpha$ - $\beta$ pruning: example 3

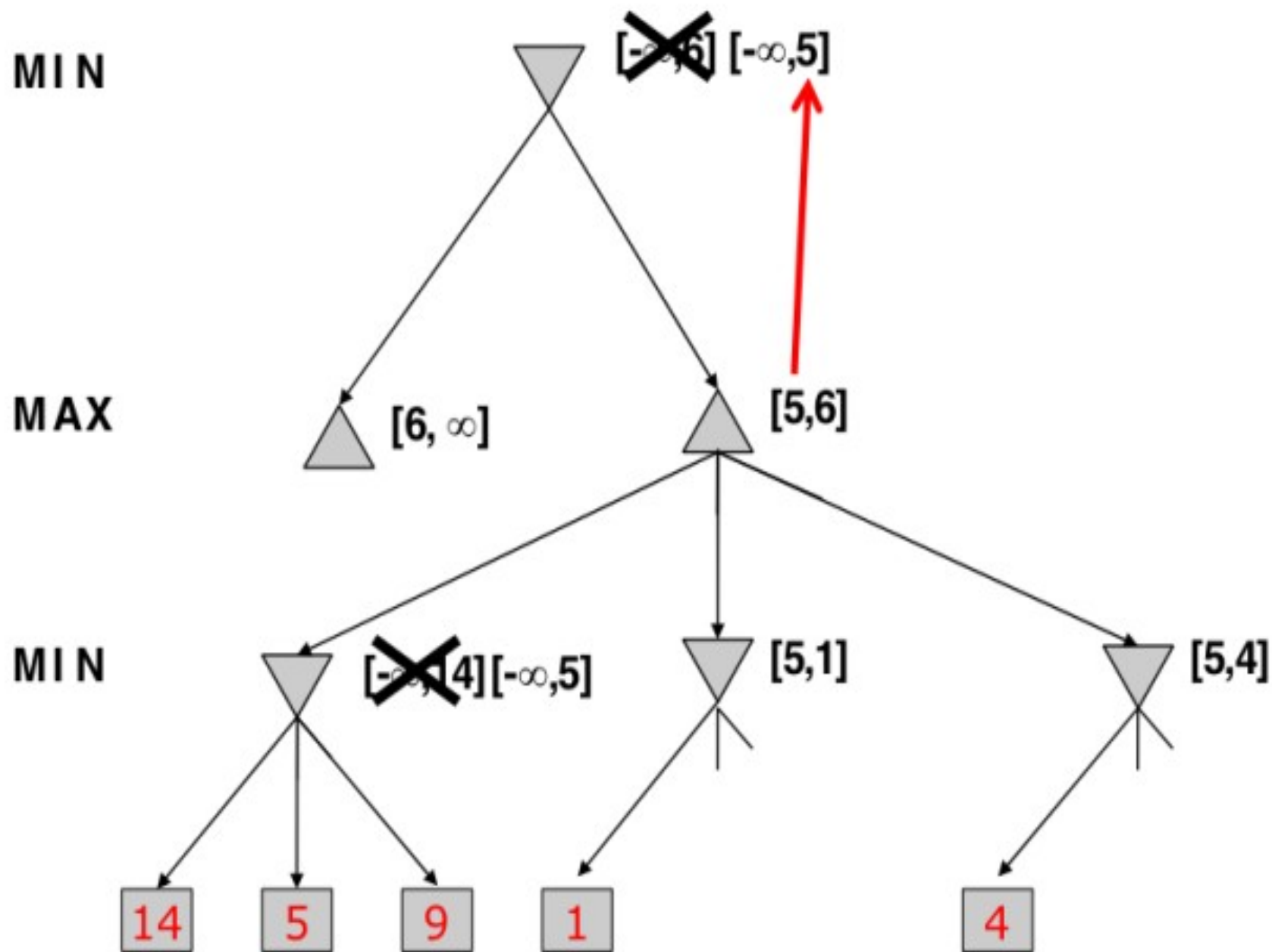


# $\alpha$ - $\beta$ pruning: example 3

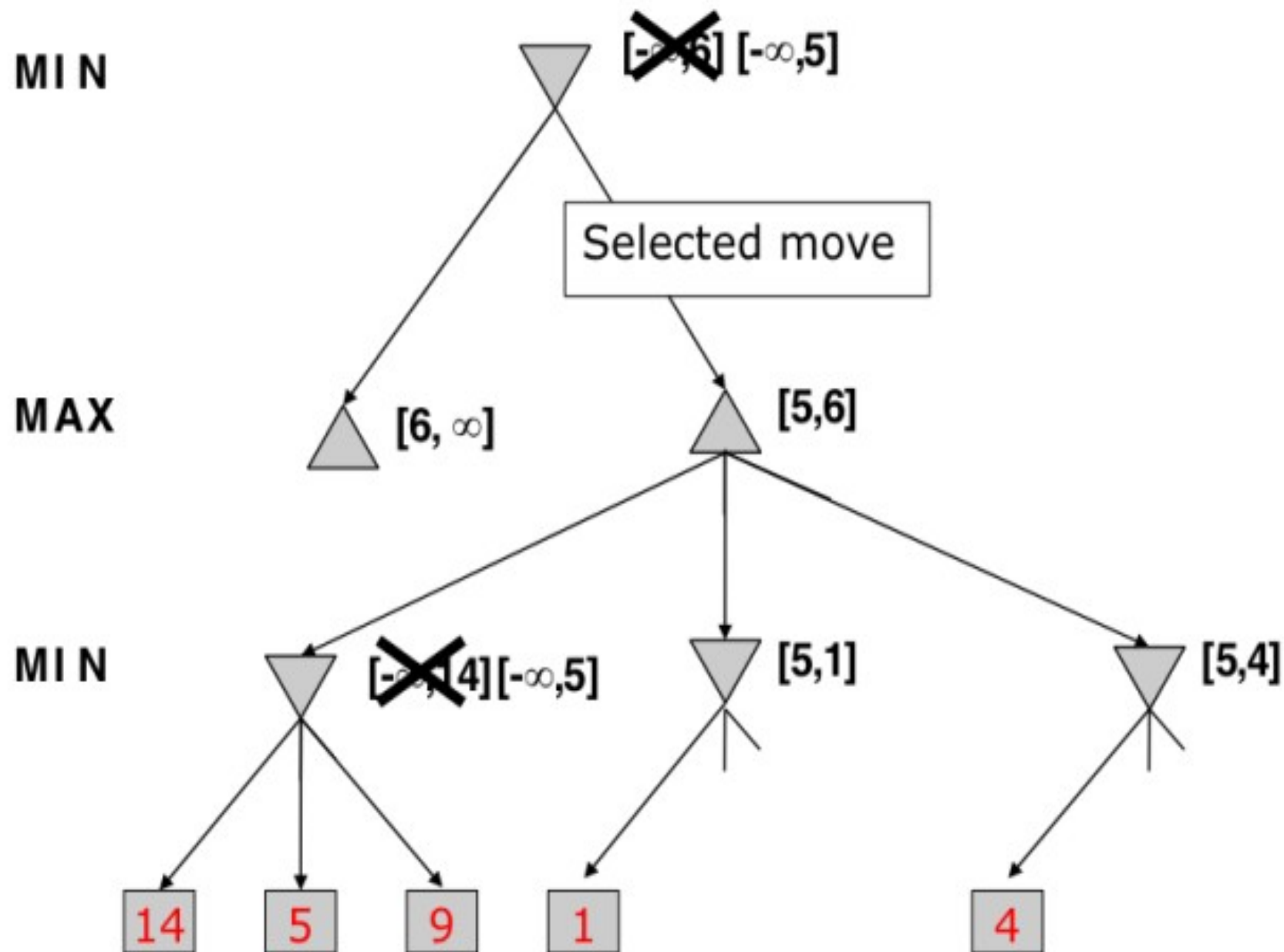




# $\alpha$ - $\beta$ pruning: example 3



# $\alpha$ - $\beta$ pruning: example 3

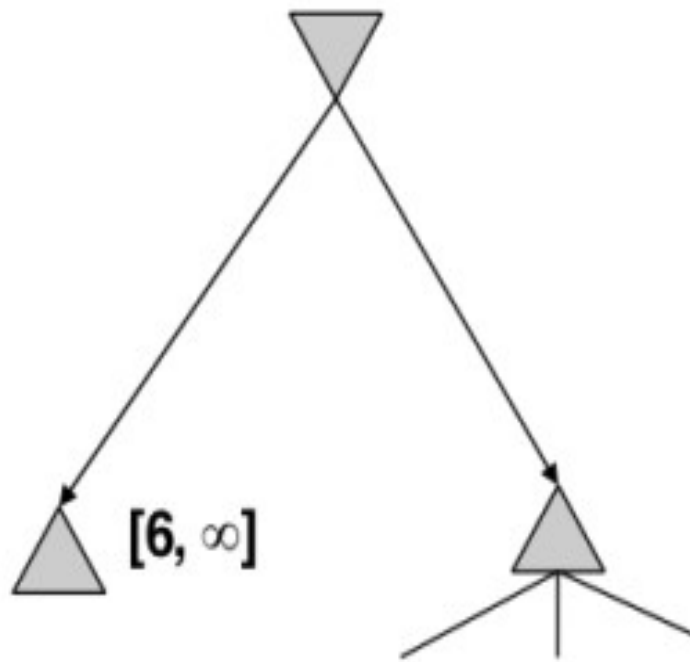


# $\alpha$ - $\beta$ pruning: example 4

MIN

MAX

MIN

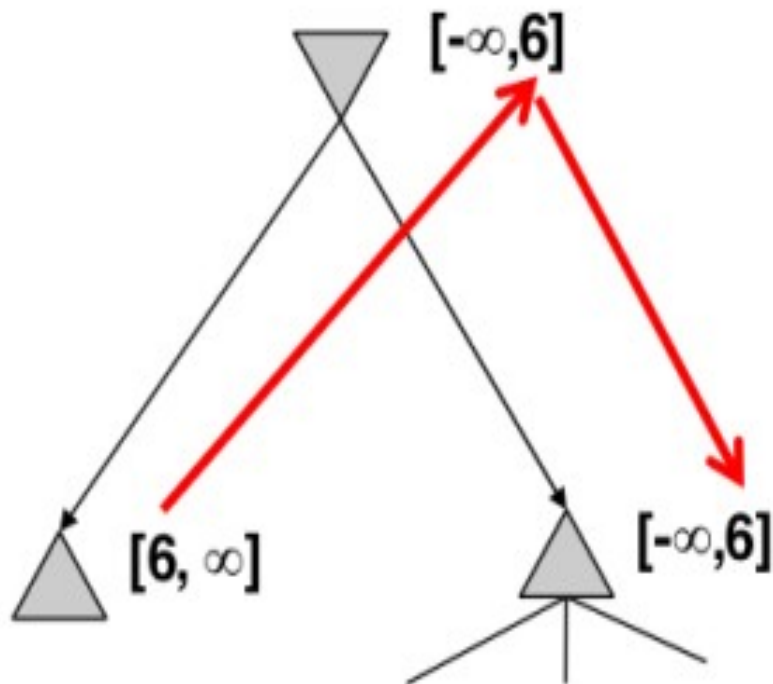


# $\alpha$ - $\beta$ pruning: example 4

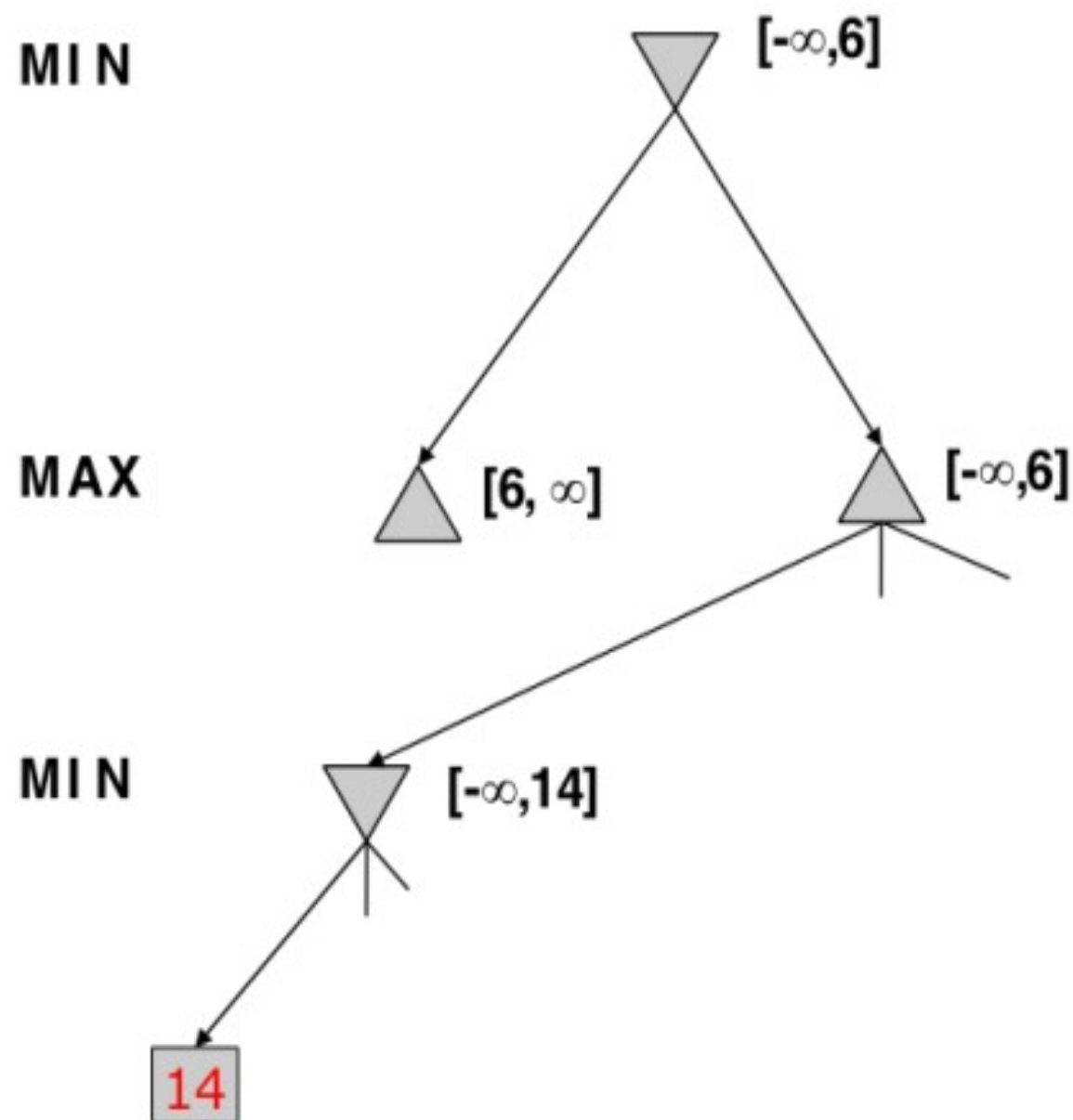
MIN

MAX

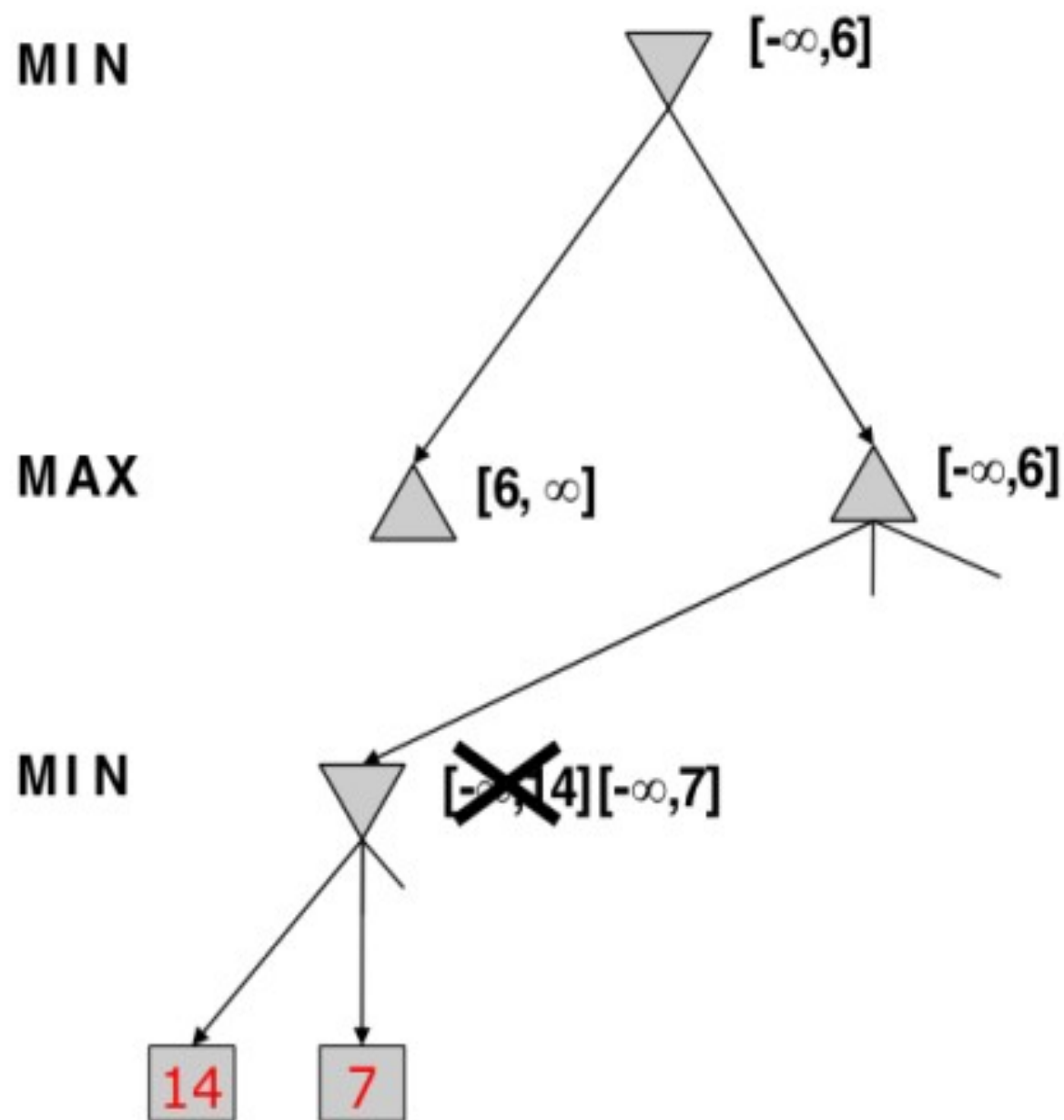
MIN



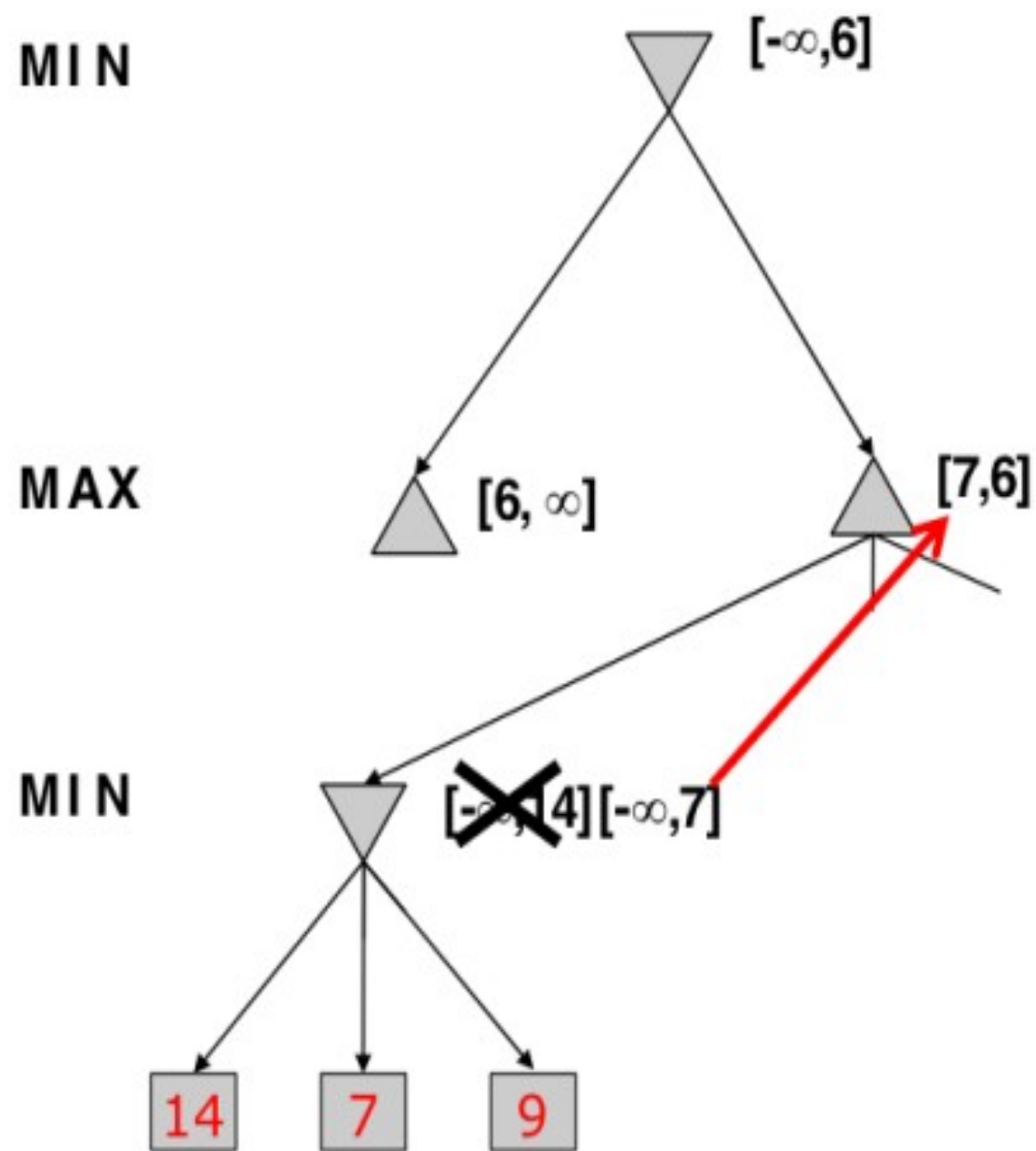
# $\alpha$ - $\beta$ pruning: example 4



# $\alpha$ - $\beta$ pruning: example 4



# $\alpha$ - $\beta$ pruning: example 4



# $\alpha$ - $\beta$ pruning: example 4

